



Belegarbeit

VARIABILITÄTSMODELLIERUNG IN KARTOGRAPHIERUNGS- UND LOKALISIERUNGSVERFAHREN

Sebastian Werner

Matr.-Nr.: 3389811, geboren am 29.08.1986

Betreut durch:

Prof. Dr. rer. nat. habil. Aßmann

und:

Dipl.-Medieninf. Christian Piechnick

Eingereicht am 25 Juli 2014

ERKLÄRUNG

Ich erkläre, dass ich die vorliegende Arbeit selbständig, unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Dresden, 25 Juli 2014

INHALTSVERZEICHNIS

1	Einleitung	9
2	Grundlagen	11
2.1	Lokalisierung	11
2.1.1	Lokale gegen globale Lokalisation	11
2.1.2	Passive und aktive Lokalisierung	12
2.1.3	Umgebung	13
2.2	SLAM	13
2.2.1	Online SLAM	14
2.2.2	Full SLAM	14
2.2.3	Offline SLAM	15
2.3	Schleifen schließen	15
2.4	Sensoren	16
2.5	Landmarken und Features	16
2.6	Kartentypen	17
2.7	Positionsschätzung	18
2.7.1	Kalman Filter	18
2.7.2	Extended Kalman Filter	19

2.7.3	Partikelfilter	20
2.7.4	Rao-Blackwellized Partikelfilter	22
2.8	Roboterbewegung	23
2.9	Zusammenfassung	24
3	Related Work	27
3.1	SLAM-Verfahren	27
3.2	Plattformen	38
3.3	Zusammenfassung	39
4	Komponenten Extraktion und Eigenschaften	41
4.1	Sensordaten aufnehmen	42
4.2	Bewegungsdaten (Odometriedaten) aufnehmen	43
4.3	Bewegungsmodell (Motion Model)	45
4.3.1	Exakte Berechnung der Position	45
4.3.2	Positionsberechnung für Partikel	46
4.4	Wahrscheinlichkeitsberechnung (Likelihood Model)	48
4.5	Kalman Filter	50
4.5.1	Motion Update	50
4.5.2	Measurement Update	52
4.6	Positionskorrektur	54
4.7	Resampling	55
4.8	Karten	57
4.9	Zusammenfassung	60
5	Feature Modell für SLAM-Verfahren	61
5.1	Feature Diagramm	61
5.2	SLAM Feature Diagramm	62
5.3	Umsetzung ausgewählter SLAM-Verfahren	64

5.4	Erweiterungsmöglichkeiten	69
5.5	Zusammenfassung	69
6	Zusammenfassung und Ausblick	71
	Literatur	74

1 EINLEITUNG

In der heutigen Zeit spielt die Automatisierung eine immer bedeutendere Rolle, speziell im Bereich der Robotik entwickeln sich immer neue Einsatzgebiete, in denen der Mensch durch autonome Fahrzeuge ersetzt wird. Dabei orientiert sich der Großteil der eingesetzten Roboter an Streckenmarkierungen, die in den Einsatzumgebungen installiert sind. Bei diesen Systemen gibt es jedoch einen hohen Installationsaufwand, was die Entwicklung von Robotersystemen, die sich mithilfe ihrer verbauten Sensorik orientieren, vorantreibt. Es existieren zwar eine Vielzahl an Robotern die dafür verwendet werden können. Die Entwicklung der Steuerungssoftware ist aber immer noch Teil der Forschung.

Für die Steuerung wird eine Umgebungskarte benötigt, an der sich der Roboter orientieren kann. Hierfür eignen sich besonders SLAM-Verfahren, die simultanes Lokalisieren und Kartographieren durchführen. Dabei baut der Roboter während seiner Bewegung durch den Raum mithilfe seiner Sensordaten eine Umgebungskarte auf und lokalisiert sich daran, um seine Position auf der Karte exakt zu bestimmen.

Im Laufe dieser Arbeit wurden über 30 verschiedene SLAM Implementierungen bzw. Umsetzungen gefunden die das SLAM Problem lösen. Diese sind jedoch größtenteils an spezielle System- bzw. Umgebungsvoraussetzungen angepasste eigenständige Implementierungen.

Es existiert keine öffentlich zugängliche Übersicht, die einen Vergleich aller bzw. des Großteils der Verfahren, z.B. in Bezug auf ihre Funktionsweise, Systemvoraussetzungen (Sensorik, Roboterplattform), Umgebungsvoraussetzungen (Indoor, Outdoor, ...), Genauigkeit oder Geschwindigkeit, gibt. Viele dieser SLAMs besitzen Implementierungen und Dokumentationen in denen ihre Einsatzgebiete, Testvoraussetzungen oder Weiterentwicklungen im Vergleich zu anderen SLAM-Verfahren beschrieben werden, was aber bei der großen Anzahl an Veröffentlichungen das Finden eines passenden SLAM-Verfahrens nicht erleichtert.

Bei einer solchen Menge an SLAM-Verfahren und Implementierungen stellen sich aus software-technologischer Sicht folgende Fragen:

1. Besteht die Möglichkeit einzelne Teile des SLAM wiederzuverwenden?
2. Besteht die Möglichkeit einzelne Teile des SLAM dynamisch auszutauschen?

Mit dieser Arbeit wird das Ziel verfolgt, diese beiden Fragen zu beantworten. Hierfür wird zu Beginn eine Übersicht über alle gefundenen SLAMs aufgebaut um diese in ihren grundlegenden Eigenschaften zu unterscheiden. Aus der Vielzahl von Verfahren werden die rasterbasierten Verfahren, welche Laserscanner bzw. Tiefenbildkamera als Sensorik verwenden, als zu untersuchende Menge ausgewählt. Diese Teilmenge an SLAM-Verfahren wird hinsichtlich ihrer nicht-funktionalen Eigenschaften genauer untersucht und versucht in Komponenten zu unterteilen, welche in mehreren verschiedenen Implementierungen wiederverwendet werden können. Anhand der extrahierten Komponenten soll ein Featurebaum aufgebaut werden, der dem Anwender einen Überblick und die Möglichkeit bereitstellt SLAM-Verfahren nach speziellen Kriterien (Systemvoraussetzungen, Umgebungen, ...) zusammenzusetzen bzw. zur Laufzeit anzupassen. Dafür müssen die verfügbaren SLAM Implementierungen und dazugehörigen Dokumentationen in Bezug auf ihre Gemeinsamkeiten und Unterschiede analysiert werden.

Um dieses Ziel zu erreichen, wird in Abschnitt 2 ein Überblick über die grundlegenden Unterteilungsmerkmale von SLAM-Verfahren bereitgestellt und die in diesem Zusammenhang zu erfüllenden Voraussetzungen erläutert.

Im Abschnitt 3 werden die ursprünglichen SLAMs aus dem Buch von Thrun et al. [TFB06] vorgestellt und einen Überblick der genauer analysierten rasterbasierten SLAM-Verfahren gegeben. Hierbei werden erste Komponenten der SLAMs, welche im Abschnitt 4 genauer mit ihrer Funktion und den dazugehörigen Implementierungsmöglichkeit besprochen werden, beschrieben.

Anhand der extrahierten Komponenten wird im Abschnitt 5 ein Feature Diagramm erstellt, mit dem einige der zuvor besprochenen SLAM-Verfahren beschrieben werden.

2 GRUNDLAGEN

Im Folgenden werden die grundlegenden Begriffe und Problemstellungen aus dem Bereich der SLAM-Verfahren kurz erklären und ein Überblick über die verschiedenen Lokalisationsarten, SLAM-Unterteilungen und Positionsschätzungs- bzw. Korrekturverfahren gegeben.

2.1 LOKALISIERUNG

Die Lokalisation wird für die Positionsbestimmung des Roboters mithilfe seiner Sensordaten auf einer existierenden Karte benötigt. Dabei kann zwischen verschiedenen Arten der Ortsbestimmung mit der *lokalen* und *globalen* Lokalisation des Roboters, bei welcher es um die Positionsbestimmung geht, unterschieden werden. Neben der Position ist aber auch die Steuerung des Roboters mit der *passiven* und *aktiven* Lokalisierung sowie die Art der Umgebung, in der sich der Roboter befindet, von Interesse. Diese verschiedenen Bereiche werden im Folgenden kurz erklären.

2.1.1 Lokale gegen globale Lokalisation

Die Lokalisation eines Roboters kann in lokale und globale Lokalisierung unterteilt werden, welche sich deutlich in ihrer Fehleranfälligkeit und dem benötigten Rechenaufwand unterscheiden. Hierbei können zwei grundlegende Typen der Lokalisierung unterschieden werden.

Die lokale Lokalisation oder auch als Positionsverfolgung bezeichnet ist der einfachste Lokalisationstyp, da das System die Startposition des Roboters auf der Karte kennt. Während sich der Roboter fortbewegt, liefern seine Sensoren Informationen über den zurückgelegten Weg und seine Umgebung. Aufgrund der Ungenauigkeiten und Toleranzen der Sensoren des Roboters, wie zum Beispiel das Zählen der Radumdrehungen, können sich Fehler während der Bewegung aufsummieren und über die zurückgelegte Strecke zu immer ungenaueren Roboterpositionen auf

der Karte führen. Die Aufgabe der Positionsverfolgung ist es, mithilfe der Sensordaten und Karteninformationen die Bewegungsfehler der Roboterhardware während der Fortbewegung auszugleichen und zu korrigieren. Die meisten Algorithmen der lokalen Lokalisierung gehen von einem sehr kleinen Fehler des Roboters aus.

Die globale Lokalisierung ist ein größeres Problem, da das System zwar eine Karte der Umgebung besitzt, jedoch keine Information darüber hat, wo sich der Roboter zum Startzeitpunkt befindet. Die Aufgabe besteht nun darin, die Roboterposition auf der Karte zu ermitteln. Hierfür wird ein hybrider Ansatz verwendet, indem mithilfe der Sensordaten alle möglichen Roboter Startpositionen auf der Karte bestimmt und diese unter Verwendung der Positionsverfolgung während der Fortbewegung des Roboters aussortiert werden. Wegen fallender Positionswahrscheinlichkeiten können immer mehr Startpositionen während der Bewegung des Roboters durch die Umgebung ausgeschlossen werden.

Das Problem des entführten Roboters nach Thrun et al. [TFB06] ist eine Erweiterung der globalen Lokalisation. In diesem Fall weiß der Roboter nichts, er kennt weder seine Position auf der Karte noch besitzt er Information darüber, ob er sich in einer für ihn bekannten Umgebung mit existierender Karte oder gar in einer komplett neuen Umgebung befindet. Laut Thrun et al. [TFB06] besteht die Relevanz dieses Problems darin, zu testen, wie gut sich ein Algorithmus zur globalen Lokalisation bei Fehlern verhält und darauf reagiert, wenn der Roboter in einer komplett unbekanntem Umgebung ausgesetzt wird.

2.1.2 Passive und aktive Lokalisierung

Nach Thrun et al. [TFB06] kann die Lokalisation in passive und aktive Lokalisation unterteilt werden. Beide Verfahren unterscheiden sich darin, ob der Lokalisationsalgorithmus für die Steuerung des Roboters zuständig ist oder nicht.

Die passive Lokalisierung ist dadurch gekennzeichnet, dass der Lokalisationsalgorithmus keinen Einfluss auf die Bewegung des Roboters hat, er erhält nur die Sensordaten und Bewegungsinformation und muss daraus die aktuelle Position schätzen. Die Robotersteuerung ist hierbei vollständig von der Lokalisation getrennt und kann von dieser nicht beeinflusst werden.

Bei der aktiven Lokalisierung bekommt der Lokalisationsalgorithmus die Steuerung über den Roboter und hat somit die Möglichkeit den Roboter zu Positionen im Raum zu bewegen, von denen er weitere Informationen benötigt. Außerdem besteht die Möglichkeit, Wege ohne Sensordaten zu vermeiden. Dies ist zum Beispiel für Laserscanner wichtig, wenn sich der Roboter weiter als die Reichweite des Lasers von Objekten wegbewegt.

2.1.3 Umgebung

Umgebungen sind Orte, die von Robotern befahren, überflogen oder ... werden und in denen sich diese lokalisieren. Dies kann zum Beispiel in Häusern, in Räumen, in Unterwasserlandschaften, im Freien, in Höhlen oder an anderen Orten sein. Sie können in statische und dynamische Umgebungen unterteilt werden, wodurch verschiedene Bedingungen an die Lokalisierungs- und Kartographierungsalgorithmen gestellt werden.

In statischen Umgebungen sind alle Objekte immer an der gleichen Position, wodurch nur Fehler aus der Roboterbewegung betrachtet und behoben werden müssen.

In dynamischen Umgebungen können mehrere Gegenstände zeitlich ihre Position ändern, verschwinden oder hinzukommen. In diesem Fall ist es für die Lokalisierung wichtig, sich an festen Objekten zu orientieren und die anderen beweglichen Objekte zu identifizieren und zu ignorieren. Bewegliche Objekte können dabei Personen, einfach zu bewegende Möbel (Stühle, Tische, ...), Türen und vieles mehr sein.

2.2 SLAM

Eines der größten Probleme in der Robotik ist das SLAM (Simultaneous Localization and Mapping) Problem, welches auch unter Concurrent Mapping and Localization (CML) bekannt ist. Die Herausforderung im SLAM besteht darin, dass der Roboter zu Beginn keine Karte der Umgebung besitzt und seine eigene Position nicht kennt. Die einzigen Informationen, die dem Roboter zur Verfügung stehen, sind Messdaten und Bewegungsinformationen.

Messdaten generiert der Roboter aus Kamerasystemen, Laserscannern oder anderen Sensoren, die auf dem Roboter montiert sind und Umgebungsinformationen relativ zu seiner aktuellen Position liefern.

Bewegungsinformationen geben an, wie sich der Roboter von einer Position der Messdatengenerierung zur nächsten Position bewegt hat. Hierbei werden Fahrabweichungen der Mechanik nicht berücksichtigt.

Die Aufgabe des SLAM Algorithmus besteht darin, während der Roboterbewegung eine Umgebungskarte aufzubauen und sich gleichzeitig an dieser zu lokalisieren. Da der Roboter seine Bewegungsinformationen besitzt, sollte davon ausgegangen werden, dass eine zusätzliche Lokalisation unnötig ist. Da aber jeder Roboterbewegung ein Fehler zugrunde liegt, muss der SLAM Algorithmus diesen Bewegungsfehler durch die simultane Lokalisierung ausgleichen.

Es gibt eine Vielzahl von SLAM Algorithmen, welche in Online-, Offline oder Full SLAM-Verfahren unterschieden werden können. Diese sind in der Abbildung 2.1 schematisch mit den Roboterpositionen x , Bewegungsinformationen u , Sensordaten z und der Karte m dargestellt. Der farbig hinterlegte Bereich markiert die während der Roboterbewegung im Speicher abgelegten Elemente. Im Folgenden werden diese SLAM Typen näher erklärt.

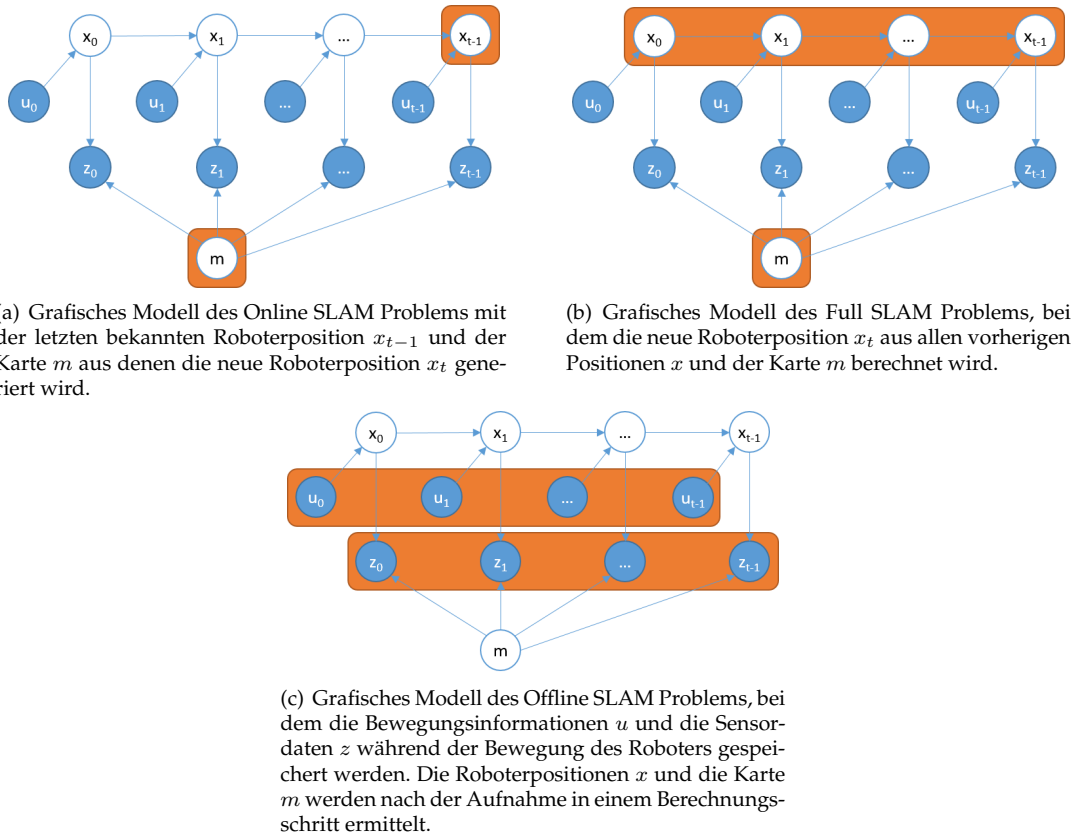


Abbildung 2.1: Grafische Modelle der verschiedenen SLAM Typen.

2.2.1 Online SLAM

Online SLAM-Verfahren speichern, wie in Abbildung 2.1(a) zu sehen, die letzte bekannte Roboterposition x_t und die dazugehörige Karte m . Anhand dieser Daten schätzt das Verfahren die neue Roboterposition x_{t+1} . Somit sind viele Algorithmen, die dem Online SLAM zugeordnet werden, inkrementelle Algorithmen, da die alten Messdaten und Positionsdaten für weitere Bewegungen nicht mehr benötigt und somit nach der Verarbeitung gelöscht werden können. Das bedeutet, dass nur immer die aktuelle Roboterposition und die dazugehörige Karte gespeichert werden, womit die Position des Roboters in Echtzeit verfolgt werden kann.

2.2.2 Full SLAM

Full SLAM-Verfahren speichern im Gegensatz zu Online SLAM-Verfahren, wie in Abbildung 2.1(b) zu sehen ist, alle Roboterpositionen x und die dazugehörige Karte m . Daraus folgt zwar, dass diese Verfahren einen erhöhten Speicherbedarf haben, jedoch besteht die Möglichkeit rückwirkend Positionen erneut zu bearbeiten bzw. zu berücksichtigen und diese Informationen in die Berechnung der nächsten Roboterposition einfließen zu lassen.

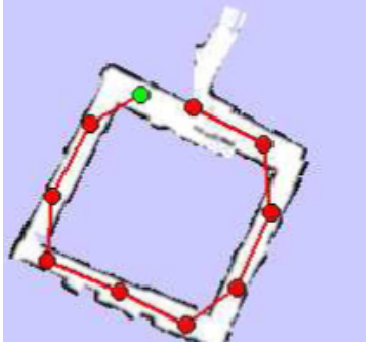


Abbildung 2.2: Karte mit einem Bewegungsfehler [Sta06]

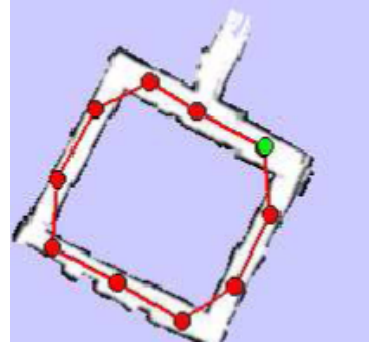


Abbildung 2.3: Karte nach dem Schleifen-schließen [Sta06]

2.2.3 Offline SLAM

Offline SLAM-Verfahren sammeln, wie in Abbildung 2.1(b), alle Sensordaten z und Bewegungsinformationen u über eine komplette Roboterbewegung. Der Vorteil dieses Verfahrens ist, dass während der Roboterbewegung nur Daten gesammelt und keine Berechnungen durchgeführt werden müssen, was wiederum bedeutet, dass während der Bewegung keine neuen Karteninformationen zur Verfügung stehen. Die Karten- und Positionsrechnung wird nach dem Sammeln durchgeführt und hat den Vorteil, dass die Karten genauer werden, da während der Berechnung auch zurückliegende Positionen verändert und damit besser an die Sensordaten angepasst werden können, wodurch Fortpflanzungsfehler im Nachhinein behoben werden können. Bei diesem Verfahren besteht die Möglichkeit während der Bewegung Karten zu berechnen, jedoch muss nach dem Eintreffen neuer Roboterpositionen die komplette Kartenerstellung erneut durchgeführt werden, was zu einem erhöhten Rechenaufwand führt.

2.3 SCHLEIFEN SCHLIESSEN

Das Schließen von Schleifen (Loop Closure) ist ein Problem, welches sich bei der Kartierung ergibt, sobald der Roboter einen Bereich der Umgebung aufnimmt, von dem er zuvor bereits eine Aufnahme gemacht hat. Da in der Bewegung des Roboters zwischen diesen Aufnahmen des gleichen Bereiches Fehler entstanden sein können, wie in Abbildung 2.2 zu sehen, führen die Gänge im oberen Bereich falsch zusammen. Damit die Aufnahmen genau aufeinander passen und eine korrekte Karte, wie in Abbildung 2.3, aufgebaut wird, muss zur Behebung dieses Fehlers ein Loop Closure Algorithmus verwendet werden. Dieser Algorithmus speichert einen Bewegungsgraphen des Roboters und prüft mit diesem, ob der Roboter einen bereits erkundeten Knoten des Graphen sieht, mit dem er noch nicht verbunden ist. Ist ein solcher Knoten gefunden, muss ein Algorithmus (wie ICP siehe Abschnitt 4.6) z.B. die Sensordaten der beiden Knoten, die verbunden werden sollen, zueinander orientieren. Anschließend muss der Fehler der letzten Roboterposition noch ausgeglichen werden. Dies kann geschehen, indem der Fehler auf alle Roboterpositionen der Schleife verteilt und die Karte dementsprechend ausgebessert wird.

Je nachdem, wie exakt die Geschwindigkeitsmessung (Odometrie) des Roboters und die Fehlerbehebung der Roboterbewegung ist, sind Loop Closure Algorithmen wichtig oder können bei sehr genauen Systemen bzw. SLAM Algorithmen außer Acht gelassen werden.

2.4 SENSOREN

Sensoren werden für die Aufnahme von umgebungsspezifischen Informationen benötigt, anhand derer Algorithmen die Kartierung und Lokalisierung durchführen. Die wichtigsten Sensoren für die SLAM Algorithmen sind Kameras (z.B. RGB-Kameras oder DRGB-Kameras), Laserscanner und Ultraschallsensoren. Genauere Erklärungen zu den Sensoren und ihren Aufnahmeformaten folgen im Abschnitt 4.1.

2.5 LANDMARKEN UND FEATURES

Bei Feature basierten SLAM-Verfahren werden Landmarken bzw. Features erkannt und für die Lokalisierung verwendet. Dabei verwendet der SLAM die bekannte bzw. wahrscheinliche Featurezuordnung um die Roboterposition anhand der Transformation zwischen den Sensoraufnahmen zu verbessern. Im Folgenden wird der Unterschied zwischen Features und Landmarken erläutert.

Features entsprechen klar identifizierbaren Objekten in der realen Welt. Dies sind in Räumen z.B. Türen, Fenster, Wände, Zimmerecken oder im Freien z.B. Bäume, Hausecken, Hydranten oder andere markante statische Objekte.

Landmarken sind Features, mit dem einzigen Unterschied, dass Landmarken für die Roboter Navigation verwendet werden und somit als virtuelle Darstellung betrachtet werden können.

In [TFB06] wird zwischen Landmarken mit bekannter Zuordnung und Landmarken mit unbekannter Zuordnung unterschieden.

Bekannte Zuordnung (Korrespondenz) bedeutet, dass Landmarken, die einmal erkannt wurden, immer wieder auch aus anderen Positionen wiedererkannt und zugeordnet werden können. Diese Zuordnung muss in einem Vorverarbeitungsschritt stattfinden, da die SLAM Algorithmen diese Informationen für ihre Berechnungen verwendet.

Eine unbekannte Zuordnung (Korrespondenz) verlangt diese Vorverarbeitung nicht und der SLAM Algorithmus versucht in seiner Abarbeitung selbst eine Zuordnung zu schätzen und handelt aufgrund dieser Schätzung. Dabei kennt der SLAM die Positionen der identifizierten Features und schätzt die Zuordnung anhand ihrer Nähe zueinander. Weswegen ein zu geringer Abstand zwischen den Features zu einer erhöhten Anzahl an Fehlzuordnungen führt, was die Positionsverbesserung verhindert.

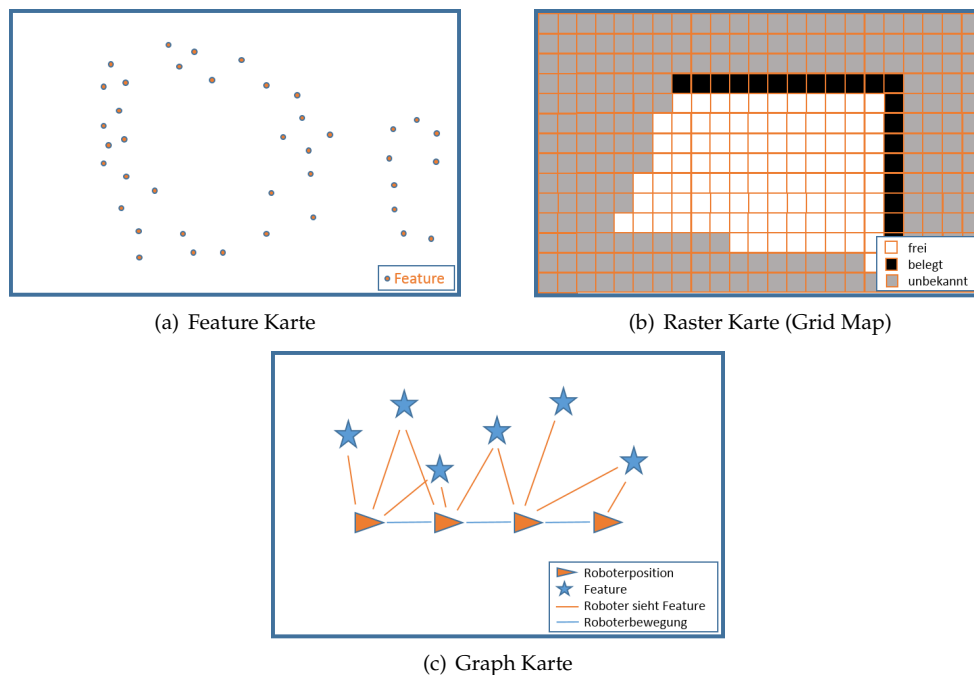


Abbildung 2.4: Diese Abbildung zeigt eine Übersicht über die drei grundlegenden Kartenrepräsentationen in SLAM-Verfahren.

2.6 KARTENTYPEN

Es kann zwischen drei allgemeinen Kartentypen (Feature, Raster und Graph basierten Karten), siehe Abbildung 2.4, unterschieden werden. Obwohl es zu jedem noch abhängig vom SLAM-Verfahren und der Implementierung mehrere besondere Formen gibt, sind diese drei die grundlegenden Typen auf denen alle basieren. Einige spezielle Kartenformen werden im Abschnitt 4.8 näher erklärt. Wohingegen im Folgenden die Hauptmerkmale der Feature, Raster (Grid) und Graph basierten Karten beschrieben werden.

Feature Karte ist eine Repräsentation in der die identifizierten Features, siehe Abbildung 2.4(a), mit ihrer Position im Kartenkoordinatensystem gespeichert werden. Die Koordinaten können abhängig von der Kartendarstellung in 2-dimensionaler oder 3-dimensionaler Form vorliegen und die Speicherdatenstruktur der Features kann den Kriterien des jeweiligen SLAM-Verfahrens angepasst werden (z.B. eine Baumstruktur).

Raster Karte (Grid Map) ist eine 2-dimensionale Darstellungsform. Wie in Abbildung 2.4(b) zu sehen, wird die abzubildende Umgebung in quadratische Felder mit auf die benötigte Genauigkeit angepasste Kantenlängen unterteilt. Es muss bei der Kantenlänge, welche größtenteils in der Literatur auf maximal 15cm begrenzt ist (z.B. nach Thrun et al. [TFB06]), auch berücksichtigt werden, dass diese kürzer als die kleinste geplante Roboterbewegung zwischen zwei Sensoraufnahmen ist, da sonst Aufnahmen irrelevant sind und zu Fehlern führen. Jedes Feld der Karte ist mit einem der Werte *frei* (*free*), *belegt* (*occupied*) oder *unbekannt* (*unknown*) belegt. Zu dieser Kartendarstellungsform gibt es noch Sonderformen, wie die Distributed Partikel Grid Map oder die Octo Map, welche im Abschnitt 4.8 erläutert werden.

Graph Karte ist eine Repräsentationsform, bei der ein Graph aufgebaut wird. Dabei gibt es eine Vielzahl von Möglichkeiten die Kanten und Knoten zu definieren, indem z.B. alle Features und deren Abstand zueinander, alle Roboterpositionen und deren Bewegungsübergänge oder anderes abgebildet werden. Die Abbildung 2.4(c) zeigt z.B. einen Graph, der aus den Roboterpositionen und den Features, die aus den jeweiligen Positionen erkannt werden, besteht. Abhängig von dem verwendeten SLAM-Verfahren können unterschiedliche Darstellungen benötigt werden. In dieser Arbeit werden Graph basierte Karten jedoch nicht näher betrachtet, da sie nicht im Fokus standen.

2.7 POSITIONSSCHÄTZUNG

Abhängig von dem verwendeten Robotersystem sind die Odometriefehler (z.B. zu weit gefahren, zu weit gedreht, ...) der Hardware unterschiedlich stark ausgeprägt. Fehler treten jedoch immer auf, auch wenn sie nur gering sind. Das Ziel der folgenden Verfahren ist es, diese Odometriefehler mithilfe der Sensordaten auf verschiedene Art und Weise zu minimieren.

2.7.1 Kalman Filter

Der Kalman Filter ist nach seinem Entwickler Rudolf E. Kálmán benannt. Das Ziel des Filters ist es Rückschlüsse auf einen Zustand anhand von fehlerbehafteten Beobachtungen zu ziehen. Die Zustandsübergänge (in diesem Fall Roboterbewegungen) sind hierbei über lineare Gleichungen definiert.

Bei Robotern wird der Filter verwendet um die Bewegungsfehler auszudrücken und somit die Genauigkeit des Systems zu ermitteln.

Der Kalman Filter besteht aus dem Mittelwert μ , der die aktuelle Position des Roboters beschreibt, und der Kovarianzmatrix Σ bestehend aus den Varianzen δ^2 . Die Kovarianz ist ein Maß für die Genauigkeit der Roboterposition, wobei die Varianzen δ^2 mit der Genauigkeit abnehmen und umgekehrt.

Die Berechnung von μ und Σ wird in zwei Teile unterteilt, das Motion Update und das Measurement Update.

Bewegungsupdate (Motion Update) beschreibt die Bewegung des Roboters. Mit jeder Bewegung wird ein Fehler mit einer definierten Varianz δ^2 dem System hinzugefügt. Dabei steigt die Varianz in der Zielposition des Roboters an.

Berechnung:

A ist die Dynamikmatrix und beschreibt die Roboterbewegung.

Q ist die Prozessrauschkovarianzmatrix mit den Standardabweichungen der Roboterbewegung.

$$\begin{aligned}\mu_t &= A_t \mu_{t-1} \\ \Sigma_t &= A \Sigma_{t-1} A^T + Q\end{aligned}$$

Das Bewegungsmodell (Motion Model) bewirkt eine Verschlechterung der Positionssicherheit.

Messwertupdate (Measurement Update) verwendet Sensorinformationen um die Verschlechterung in dem Motion Model wieder auszugleichen, indem die Kovarianzmatrix Σ und die mittlere Roboterposition μ verbessert werden. Hierbei wird aber auch ein kleiner Fehler wieder durch die Ungenauigkeit des Sensors hinzugefügt, der zu berücksichtigen ist.

Dabei wird ein Kalman Gain K bestimmt, der Aufschluss darüber gibt, ob die Messwerte oder die Bewegung vertrauenswürdiger sind. Je kleiner der Gain K , desto mehr entsprechen die Messwerte den Bewegungswerten. Sollten die Messwerte etwas völlig anderes aussagen, werden die Elemente in K größer.

Der Systemzustand wird folgendermaßen aktualisiert:

R ist die Kovarianzmatrix mit dem Rauschen der Messdaten.

C ist die Messmatrix.

z sind Messdaten.

I ist eine Einheitsmatrix.

$$\begin{aligned}K &= \Sigma C^T (C \Sigma C^T + R)^{-1} \\ \mu &= \mu + K(z - C\mu) \\ \Sigma &= (I - KC)\Sigma\end{aligned}$$

2.7.2 Extended Kalman Filter

Der Extended Kalman Filter ist die Weiterentwicklung des Kalman Filters für die Verwendung von nichtlinearen Zustandsübergängen (z.B. Sinus, Cosinus, ...). Dabei muss um den aktuellen Mittelwert und die aktuelle Kovarianz linearisiert werden. Das bedeutet, dass die Funktion an der aktuellen Position partiell abgeleitet wird. Dies geschieht, indem die Jacobi-Matrix gebildet und mit dieser weiter gerechnet wird.

Die Modelle ändern sich folgendermaßen:

Motion Update Berechnung:

$g_A(\mu, u)$ berechnet die neue Roboterposition mit dem Bewegungsvektor u .

J ist die Jacobi-Matrix der Dynamikmatrix von der Roboterbewegung.

Q ist die Prozessrauschkovarianzmatrix mit den Standardabweichungen der Roboterbewegung.

$$\mu_t = g_A(\mu, u)$$

$$\Sigma_t = J\Sigma_{t-1}J^T + Q$$

Measurement Update Berechnung:

R ist die Kovarianzmatrix mit dem Rauschen der Messdaten.

J ist die Jacobi-Matrix der Messmatrix.

z sind Messdaten.

I ist eine Einheitsmatrix.

$$K = \Sigma J^T (J\Sigma J^T + R)^{-1}$$

$$\mu = \mu + K(z - g_c(\mu))$$

$$\Sigma = (I - KJ)\Sigma$$

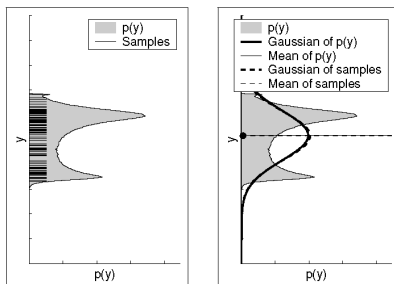


Abbildung 2.5: Das linke Bild zeigt die Wahrscheinlichkeitsverteilung für die Roboterpositionen und die dazugehörigen Partikel (Samples) gehäuft an den Extrempunkten der Funktion. Das rechte Bild zeigt die entstehende Gaußkurve für den EKF mit dem Mittelwert zwischen den Extrempunkten [TFB06].

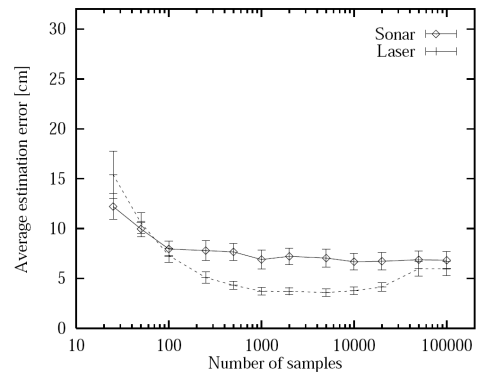


Abbildung 2.6: Diagramm mit dem durchschnittlichen Positionsfehler bei Laser- und Ultraschallsensoren nach Fox et al. [FTBD].

2.7.3 Partikelfilter

Der Partikelfilter ist eine parameterfreie Implementierung eines Bayes-Filters, der als Alternative für einen parametrisierten Gauß-Filter verwendet werden kann. Die Aufgabe des Partikelfilters besteht darin eine konkrete Zielmenge an Roboterpositionen aus einer wahrscheinlich generierten Menge zu extrahieren. Nach Fox et al. [FTBD] wird für Laserscanner, wie in der Abbildung 2.6 dargestellt, eine Partikelanzahl von 1000-5000 für einen möglichst kleinen Fehler benötigt. Der Vorteil des Partikelfilters ist, dass eine Menge an Beispielrepräsentationen existiert, die jeweils eine konkrete Position zu einer definierten Zeit beschreiben. Somit kann im Vergleich zu einem parametrisierten Gauß ein größerer Raum an möglichen Positionen abgedeckt werden.

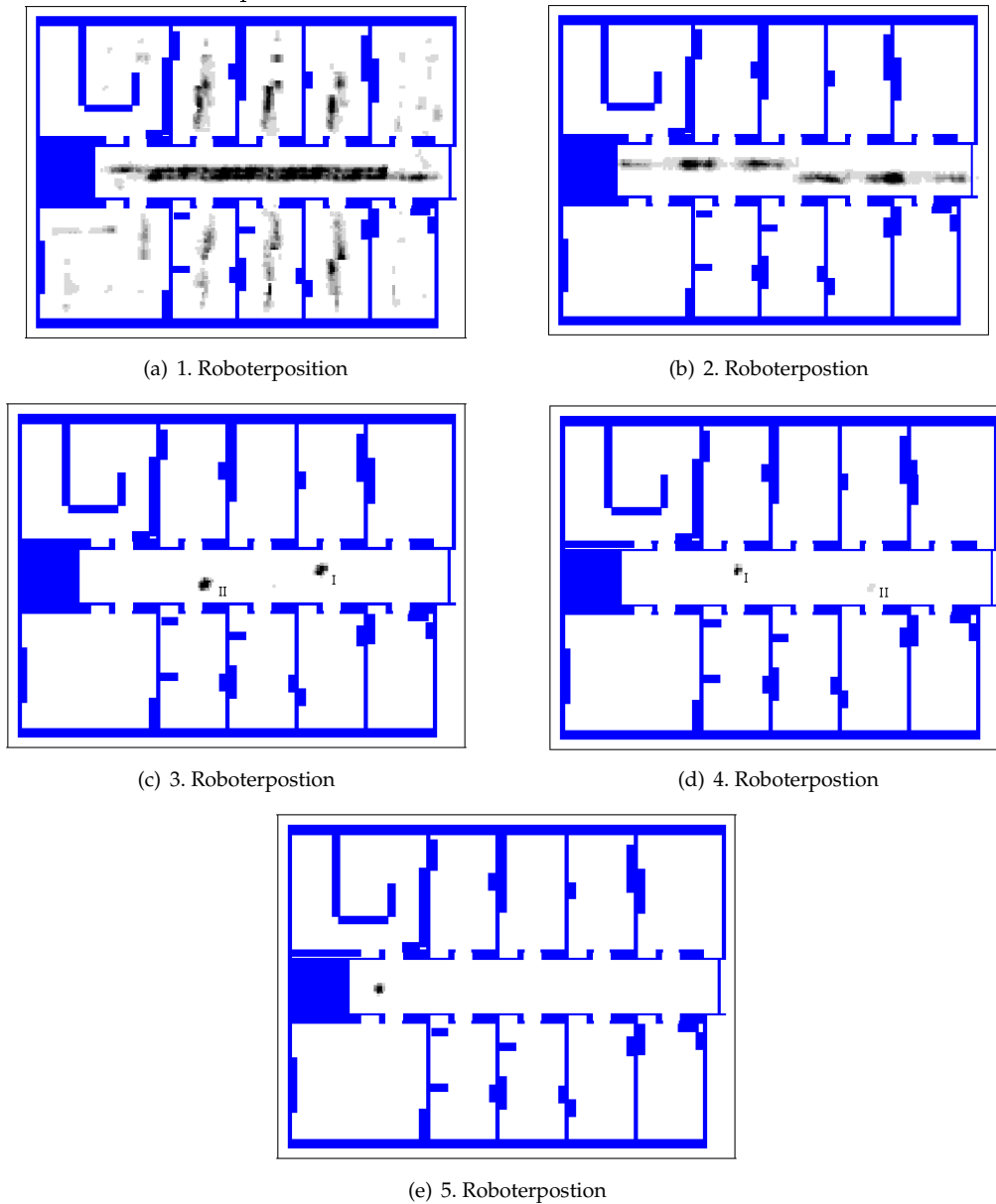


Abbildung 2.7: In den Grundriss Abbildungen von Thrun et al. [TFB06] sind alle möglichen Roboterpositionen eingezeichnet. Mit der Bewegung des Roboters und den aufgenommenen Sensorinformationen werden mit jeder weiteren Roboterposition die möglichen Positionen des Roboters auf dem Grundriss weniger.

In Abbildung 2.7 ist eine globale Lokalisierung mithilfe des Partikelfilters dargestellt. Dabei sind in der Abbildung 2.7(a) alle möglichen Roboterpositionen in der Karte eingezeichnet, an denen sich der Roboter laut seiner Laserscanner Aufnahme befinden kann. Während sich der Roboter in den Abbildungen 2.7(b) bis 2.7(e) weiterbewegt, werden es immer weniger mögliche Positionen, da die Sensorinformationen jeweils weitere Partikel ausschließen. In diesem Beispiel sind nach 4 Roboterbewegungen genug Informationen gesammelt, um alle Partikel auf einen kleinen Bereich der Umgebungskarte zu beschränken.

Das durch den Partikelfilter verfolgte Ziel ist, eine Liste von Positionen mit ihren Wahrscheinlichkeiten so zu bearbeiten, dass sich deren Anzahl nicht ändert, aber die Liste nach dem Abarbeiten überwiegend mit den wahrscheinlichsten Positionen belegt ist. Hierfür findet eine Filterung statt, die einige Positionen mit geringer Wahrscheinlichkeit entfernt und dafür Positionen mit höherer Wahrscheinlichkeit mehrfach verwendet. Somit beinhaltet die Partikelliste nach dem Filtern genauso viel Elemente wie zuvor.

Abbildung 2.5 zeigt die Vorteile des Partikelfilters im Vergleich zu einem Gauß-Filter. Wie auf dem linken Bild zu sehen ist, gibt es mehrere Maximalpunkte an denen sich der Roboter befinden kann. Es häufen sich die Partikel an den Extrempunkten und treten in den restlichen Bereichen vereinzelt auf. Im rechten Bild sind die Gaußverteilung und der Mittelwert, der sich zwischen den beiden Extrempunkten ansiedelt, dargestellt. Der Partikelfilter hat den Vorteil, dass im Vergleich zum Gauß-Filter alle Bereiche abgedeckt sind und der Algorithmus keine Entscheidung für eine wahrscheinlichste bzw. durchschnittliche Position treffen muss. Somit können beide Extrempunkte aus Abbildung 2.5 in den nächsten Berechnungsschritt übernommen werden. Zusätzlich werden auch noch einige Positionen mit niedriger Wahrscheinlichkeit weiterverwendet, womit verhindert wird, dass sich der Roboter auf eine falsche Bewegung (z.B. aufgrund einer fehlerhaften Sensorinformation) fokussiert.

Ablauf eines Partikelfilters:

1. Initialisieren aller Partikel
2. Berechnen des nächsten Partikelfilter Zeitstempels
 - 2.1. Zu jedem Partikel die Folgeposition schätzen
3. Berechnen der Partikelwahrscheinlichkeiten (Gewichte)
4. Resampling durchführen
5. Für den nächsten Zeitstempel wieder zu 2. gehen

2.7.4 Rao-Blackwellized Partikelfilter

Der Rao-Blackwellized Partikelfilter ist ein Partikelfilter mit der Erweiterung, dass ein Teil des Zustandes analytisch berechnet wird. Dafür werden die berechneten Partikelpositionen mithilfe des Extended Kalman Filters analysiert und deren Varianz verringert, wodurch die Positionen schon vor dem Resampling verbessert werden. In dem Rao-Blackwellized Partikelfilter werden die Vorteile des Extended Kalman Filters und des Partikelfilters vereint. Das ist einerseits die Möglichkeit mehrerer Extrempunkte in der Positionsbestimmung zuzulassen und andererseits diese an den Extramalpositionen durch den Extended Kalman Filter auszurichten.

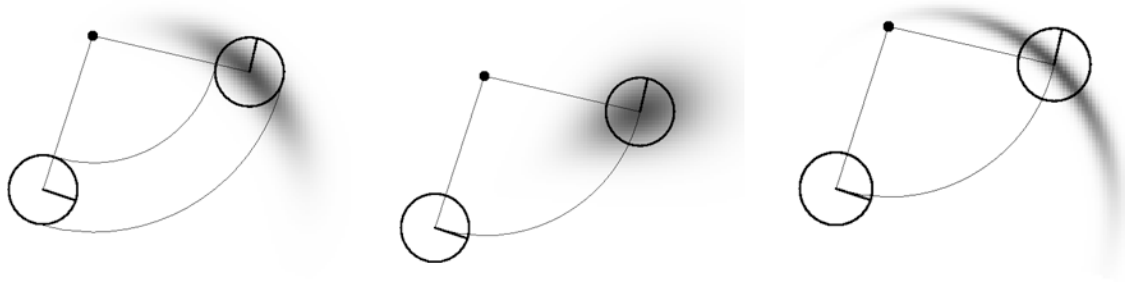


Abbildung 2.8: Velocity Bewegungsmodell mit verschiedenen Fehlerparametern [TFB06]

Weil die Partikel somit exakter sind, werden weniger Partikel für dieselbe Genauigkeit benötigt. Nach Grisetti et al. [GSB] sind 100 Partikel ausreichend, in einigen Beispielumgebungen reichen sogar 15 bis 30 Partikel für die gleiche Genauigkeit aus.

Ablauf eines Rao-Blackwellized Partikelfilters:

1. Initialisieren alle Partikel
2. Berechnen des nächsten Partikelfilter Zeitstempels
 - 2.1. Kalman Filter mit Hilfe der Messwerte aktualisieren
 - 2.2. Zu jedem Partikel die Folgeposition schätzen
 - 2.3. Kalman Filter Messwerte und Zeit aktualisieren
3. Berechnen der Partikelwahrscheinlichkeiten (Gewichte)
4. Resampling durchführen
5. Für den nächsten Zeitstempel wieder zu 2. gehen

2.8 ROBOTERBEWEGUNG

In dem Buch [TFB06] werden zwei Bewegungsmodelle für Roboter betrachtet, das Velocity und das Odometry Modell. Im Abschnitt 4.2 erfolgt eine genauere Erklärung dieser beiden Modelle.

Im Velocity Modell wird die Bewegung des Roboters kreisförmig um einen Punkt durch einen Bewegungs- und Drehungswert beschrieben. Dieses Modell wird in dem Buch [TFB06] bevorzugt und hauptsächlich verwendet. Wahrscheinlich weil das Modell für beide Roboterbewegungsarten aus dem Abschnitt 4.2 eingesetzt werden kann.

Im Odometrie Modell wird davon ausgegangen, dass sich der Roboter zu Beginn seiner Bewegung um einen Winkel dreht, danach eine Strecke geradeaus fährt und sich dann wieder dreht. Dieses Modell kann nur für Roboter mit getrennt voneinander auf einer Achse angetriebenen Rädern aus dem Abschnitt 4.2 verwendet werden.

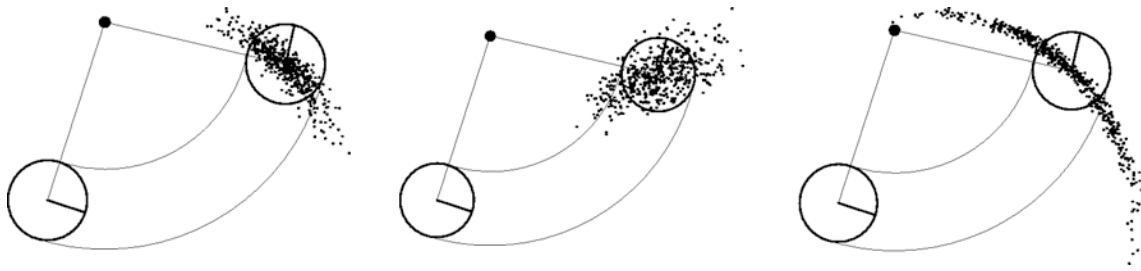


Abbildung 2.9: Velocity Bewegungsmodell mit verschiedenen Fehlerparametern als Partikeldarstellung [TFB06]

Bei der Fehlerbetrachtung in der Roboterbewegung ist zu erkennen, dass abhängig von der Fehlerbetrachtung des Roboters verschiedene Zielflächen für den Roboter entstehen. Die Abbildung 2.8 zeigt mehrere kreisförmige Velocity Modell Bewegungen von Robotern mit verschiedenen Fehlerparametern. Hierbei beschreiben die schwarzen Flächen die Position des Roboters nach der Bewegung. Anhand der Graustufung der Zielposition, die durch einen Gauß-Filter berechnet werden kann, ist die Wahrscheinlichkeit für die verschiedenen Roboterpositionen zu erkennen. Je dunkler ein Bereich, desto wahrscheinlicher ist es, dass der Roboter nach der Bewegung dort steht. In der linken und mittleren Bewegung ist eine Kombination aus Rotations- und Bewegungsfehler dargestellt, mit dem Unterschied, dass bei dem linken ein höherer Rotationsfehler und im mittleren ein höherer Bewegungsfehler existiert. Das rechte Bild stellt einen großen Rotationsfehler mit nur minimalem Bewegungsfehler dar.

Die Abbildung 2.9 zeigt Beispiele für Partikelverteilungen, hierbei sind die Fehler ebenso wie in Abbildung 2.8 verteilt. Der Unterschied ist aber, dass bei Partikelfiltern keine Gauß-Filter verwendet, sondern nur eine Vielzahl von möglichen Positionen berechnet werden, die in ihrer Häufung mit abnehmender Wahrscheinlichkeit weniger werden. Durch die Partikel Darstellung besteht die Möglichkeit eine Vielzahl verschiedener Roboterpositionen zu berechnen, an denen der Roboter aufgrund seiner Fehlerparameter nach der Bewegung stehen kann.

2.9 ZUSAMMENFASSUNG

In diesem Kapitel wurde eine Überblick über die wichtigsten Begriffe und Unterteilungs- bzw. Unterscheidungsmöglichkeiten der SLAM-Verfahren gegeben sowie diese kurz erläutert. Die folgenden Kapitel bauen auf den Begriffen dieses Abschnitts auf und werden im Bezug auf die relevanten Bereiche in der weiteren Arbeit präzisiert und genauer erläutert.

Mithilfe der eingeführten SLAM Typen (online, full und offline) werden die SLAM-Verfahren im Abschnitt 3.1 eingeordnet. Weiterhin wurden die verschiedenen Sensoren und Kartenrepräsentationen im allgemeinen aufgezeigt, damit die grundlegenden Arten für die folgenden Abschnitte bekannt sind, in welchen auf einige Fakten noch näher eingegangen und deren Vor- bzw. Nachteile sowie weitere Unterteilungsmöglichkeiten aufgezeigt werden. Das wird z.B. im Bereich der Raster Karten im Abschnitt 4.8 genauer erfolgen.

Neben den verschiedenen Bestandteilen (Extended Kalman Filter, Partikel Filter ...) und Unterteilungsmöglichkeiten wurden in diesem Abschnitt auch viele Merkmale benannt, die ein SLAM-Verfahren erfüllen muss bzw. die bei der Konzeptionierung und dem Bau eines SLAM-Algorithmus

mus bedacht werden müssen. Das betrifft sowohl die Bedingungen mit statischen und dynamischen Umgebungen als auch die benötigten Lokalisationsverfahren abhängig vom Anwendungsfall, ob z.B. eine globale Lokalisierung oder ein entführter Roboter in dem Einsatzszenario benötigt werden.

Dieser Abschnitt gibt einen Überblick über die für diese Arbeit wichtigsten SLAM Begriffe, jedoch sind damit noch nicht alle relevanten Fakten zum Erstellen und Arbeiten mit SLAM-Verfahren aufgezeigt.

3 RELATED WORK

Es gibt eine Vielzahl verschiedener SLAM-Verfahren und einige Plattformen, die SLAM-Algorithmen bereitstellen bzw. in ihren Systemen verwenden. Im Folgenden wird eine Übersicht über alle gefundenen SLAM-Verfahren gegeben und davon die grundlegenden Verfahren aus dem Buch von Thrun et al. [TFB06] kurz erklärt. Danach werden ausgewählte Grid Map basierte Verfahren in ihrer Funktionsweise erläutert. Am Ende gibt es einen Überblick über einige Plattformen bzw. Frameworks, die SLAM-Verfahren implementieren.

3.1 SLAM-VERFAHREN

SLAM-Verfahren beschreiben ein großes Forschungsfeld, aus dem meines Wissens momentan noch keine kommerziellen Produkte hervorgegangen sind. Jedoch gibt es einen großen Anwendungsbereich für diese Verfahren, der sich stark in den verschiedenen System- und Umgebungsvoraussetzungen unterscheidet. Es können z.B. verschiedene Umgebungen grob in Indoor, Outdoor, unter Wasser und in der Luft unterschieden werden, wobei diese wiederum unterteilt werden können, was im Fall der Innenbereiche, z.B. einzelne Räume, Flure oder eine Mischung von beidem sein kann. Im weiteren können auch Bergwerke, eingestürzte Gebäude oder andere Umgebungen, die schwer zugänglich und einen anderen Aufbau bieten, dazu gezählt werden. Wenn nun weiter unterteilt wird, ist das eine große Anzahl verschiedener Umgebungen in denen SLAM-Verfahren zum Einsatz kommen können. Hinzu kommen noch andere Kriterien, wie z.B. die Systemvoraussetzungen, wobei die Rechenleistung und der Speicher des Computers berücksichtigt werden müssen. Gleichwohl es in den letzten Jahren zu großen Fortschritten im Bereich der Prozessoren und Speicher gekommen ist, führt die hohe Komplexität der Berechnungsschritte aktueller SLAM-Algorithmen dazu, dass sie nur schwer in Echtzeit berechenbar sind. Hierfür gibt es Arbeiten wie z.B. von Engelhardt [Eng], bei der das Ziel der Echtzeitfähigkeit verfolgt wird.

Die zur Verfügung stehende Sensorik ist ein wichtiger Anpassungspunkt. Noch vor einigen Jahren setzten die Anwender hauptsächlich RGB-Kameras für ca. 100Euro als preiswerte Sensorik im Vergleich zu professionellen Laserscannern für einige 1000Euro, die hauptsächlich von Uni-

versitäten oder anderen Forschungseinrichtungen verwendet wurden, ein. In den letzten Jahren kamen preiswerte Tiefenbildkameras als Laserscanner Alternative, wie die Kinect¹ von Microsoft auf den Markt, was die Entwicklung von SLAM-Algorithmen stark angetrieben hat. Hieraus resultiert, dass viele der Algorithmen auf die neueren Sensoren angepasst wurden und sich Verfahren speziell für neuere Kameras entwickelt haben.

Aus dieser Vielzahl an Unterscheidungsmerkmalen ergeben sich viele Ziele, die verfolgt werden müssen, wodurch eine große Anzahl verschiedener Forschungsrichtungen entsteht.

Tabelle 3.1: SLAM-Verfahren

Algo- rithmus	Kartentyp	Sensor	Feature De- tektor	Umgebung	Sprache
EKF-SLAM [TFB06]	Feature Map	Laser, Kame- ra	extern	Indoor, Out- door	beliebig
Graph- SLAM [TFB06]	Graph Map	Laser, Kame- ra	extern	Indoor, Out- door	beliebig
SEIF-SLAM [TFB06]	Graph Map, Information Matrix	Laser, Kame- ra	extern	Indoor, Out- door	beliebig
FastSLAM 1.0 [TFB06]	Feature Map, Grid Map	Laser, Kame- ra	extern, kei- nen	Indoor, Out- door	beliebig
FastSLAM 2.0 [TFB06]	Feature Map, Grid Map	Laser, Kame- ra	extern, kei- nen	Indoor, Out- door	beliebig
CEKF-SLAM [HCHX12]	Feature Map	Kamera	extern	Indoor ,Out- door	Matlab
EKFmono SLAM [ope]	Feature Map	Kamera	extern	Outdoor	Matlab
iSAM SLAM [KMRD08]	Feature Map, Pose Graph	Laser	extern	Outdoor	c++, OCaml
iSAM2 SLAM [KJR ⁺]	Feature Map, Graph Map	Laser	extern	Indoor, Out- door	keine Anga- ben
Square Root SLAM [KDa]	Feature Map, Pose Graph	Kamera	extern	Indoor	keine Anga- ben
Fast In- cremental Square Root Informati- on Smoo- thing [KRD]	Feature Map, Pose Graph	beliebig	extern	Outdoor	keine Anga- ben
Loopy SLAM [RKD]	Feature Map, Pose Graph	beliebig	extern	Outdoor	keine Anga- ben

¹<http://www.microsoft.com/en-us/kinectforwindows/> und <http://de.wikipedia.org/wiki/Kinect> vom 18.06.2014

Tabelle 3.1: SLAM-Verfahren

Algo- rithmus	Kartentyp	Sensor	Feature De- tektor	Umgebung	Sprache
D-SLAM [WHD]	Information Matrix	Laser	extern	Indoor	Mathlab
Linear SLAM [ZHD]	Feature Map, Pose Graph	beliebig	extern	Indoor, Out- door	Matlab, c++
HOG-Man SLAM [GKS ⁺]	Graph Map	beliebig	extern	Indoor, Out- door	c++
RatSLAM [ope]	Topological Map	Kamera	extern	Indoor, Out- door	c++
RT-SLAM [RGS ⁺]	Feature Map	Kamera	extern	Indoor	c++
Java Fast- SLAM ²	Feature Map	beliebig	extern	Outdoor	Java
SLAM++ [SMNS ⁺]	Feature Map, Sparse Matrix	RGBD- Kamera	extern (3D Objekte)	Indoor	c++
SPA-SLAM [KGK ⁺]	Sparse Ma- trix	Laser	keinen	Indoor	c++
Vector Field SLAM [GEFM12]	Feature Map, Sparse Matrix	Strahlungs- messgerät, Wifi	keinen	Indoor	keine Anga- ben
ESEIF-SLAM [WEL]	Graph Map, Information Matrix	beliebig	extern	Indoor, Out- door, unter Wasser	keine Anga- ben
g2o-SLAM [KGS ⁺]	Graph Map	beliebig	extern	Indoor, Out- door	c++
Range- Only SLAM [BFMG08]	Feature Map, Grid Map	Laser	keinen	Indoor, Out- door, unter Wasser	c++
bearing-only SLAM [KDb]	Feature Map	Kamera	extern	Indoor	keine Anga- ben
Gmapping [ros]	Grid Map	Laser	keinen	Indoor	c++
DP-SLAM [EP]	Distributed Particle Grid Map	Laser	keinen	Indoor	c++
RBPF-SLAM [SG08]	Grid Map	Laser	keinen	Indoor, Out- door	c++

²<http://www.oursland.net/projects/fastslam/> vom 24.07.2014

Tabelle 3.1: SLAM-Verfahren

Algo- rithmus	Kartentyp	Sensor	Feature De- tektor	Umgebung	Sprache
ICP-SLAM [TOA ⁺]	Grid Map, Pose-Sensor Graph	Laser	keinen	Indoor	c++
Grid- SLAM [ope]	Grid Map	Laser	keinen	Indoor	c++
Tiny-SLAM oder Core- SLAM [SH]	Grid Map	Laser	keinen	Indoor	c++
Online 6D SLAM [EHE ⁺]	Octo Map	D-Kamera	keinen	Indoor	c++
SLAM 6D [ope]	Octo Map	D-Kamera	keinen	Indoor	c++
Dense Visual SLAM [KSC]	Pose-Sensor Graph	RGBD- Kamera	extern	Indoor	c++
trajectory- oriented EKF SLAM [GCB ⁺ 09]	Pose-Sensor Graph	Radar Sensor	keinen	Outdoor, un- ter Wasser	Matlab, c++
Hector SLAM [KMG ⁺]	Grid Map	RGBD- Camera	optional, kei- nen	Indoor	c++
Octo SLAM [FHC ⁺]	Octo Map	2D-Laser	keinen	Indoor, Out- door	c++
6D SLAM [ros]	Octo Map	2D-Laser, D- Camera	keinen	Outdoor	c++
SeqSLAM [MW12]	Image Graph	Camera	extern	Outdoor	Matlab
FootSLAM [BR11]		Pedometer	keinen	begehbar durch Men- schen	keine Anga- ben
PlaceSLAM [BR11]	Feature Map	Human Input	keinen	begehbar durch Men- schen	keine Anga- ben
WiSLAM [BR11]		Pedometer, Wifi	keinen	begehbar durch Men- schen	keine Anga- ben

Während der Recherchen wurde eine große Anzahl verschiedener Algorithmen gefunden, wobei sich einige kaum unterscheiden, jedoch unter verschiedenen Namen veröffentlicht sind. Die Tabelle 3.1 gibt eine Übersicht der gefundenen SLAM Implementierungen und deren allgemeine Eigenschaften wieder. Dabei wird nach dem Kartentyp, dem Sensor, ob ein Feature Detektor be-

nötigt wird, der Umgebung und der Programmiersprache unterschieden. Hierbei wurde zu den gefundenen SLAM-Verfahren je nach Verfügbarkeit die Implementierung oder der Paper analysiert und die Unterscheidungsmerkmale identifiziert. Es gibt jedoch nicht zu jedem SLAM eine frei zugängliche Implementierung bzw. Angaben zur Programmiersprache, weswegen bei einigen Verfahren dazu keine Aussage existiert. Es ist eindeutig zu erkennen, dass der Großteil in `c++` bzw. *Matlab* implementiert ist.

Bezüglich der Kartentypen werden Feature Map, Pose Graph, Graph Map, Information Matrix und Sparse Matrix in dieser Arbeit nicht näher beschrieben, wohingegen Grid Map, Octo Map und Distributed Particle Grid Map im Abschnitt 4.8 erklärt werden. Die verschiedenen Sensortypen mit Ausnahme des Strahlungsmessgerätes, Radars und Schrittzählers werden im Abschnitt 4.1 beschrieben, jedoch ist bei einigen Verfahren in der Tabelle *beliebig* angegeben, in diesen Fällen wurden Beispieldatensätze zum Testen verwendet in denen keine Aufnahme bzw. Vorverarbeitung z.B. durch einen Feature Detektor mehr nötig war. Bei den Einsatzgebieten der SLAM-Verfahren haben sich drei verschiedene Umgebungen mit Indoor (in geschlossenen Räumen, Bergwerken ...), Outdoor und Unterwasser ergeben, in die die verschiedenen Verfahren aufgrund ihrer getesteten Einsatzszenarien eingeteilt werden können.

Im Folgenden werden repräsentative SLAM-Verfahren näher betrachtet und ihre Besonderheiten aufgezeigt. Beginnend mit den grundlegenden SLAM-Verfahren (EKF-SLAM, Graph-SLAM, SEIF-SLAM und FastSLAM) aus [TFB06] wird ein Überblick über die SLAM-Verfahren gegeben auf denen ein Großteil der oben aufgelisteten Verfahren basiert. Dabei werden die Grundlagen und die Unterscheidungsmerkmale in ihrer Herangehensweise betrachtet.

In dem Buch [TFB06] gehen die Autoren von dem Vorhandensein von Landmarken aus und unterscheiden hauptsächlich zwischen Landmarken mit bekannter und unbekannter Zuordnung in ihren Beispielen. Erst für den FastSLAM Algorithmus wird ein Beispiel unter Verwendung von Laser Sensordaten und einer Occupancy Grid Map betrachtet.

Nachdem ein Überblick über die grundlegenden Algorithmen in den vier SLAM Varianten gegeben ist, wird das Hauptaugenmerk auf den Occupancy Grid Map basierten Verfahren liegen. Die anderen SLAM Verfahren können in weiteren Forschungsarbeiten in Bezug auf meine Ziele betrachtet werden.

EKF-SLAM (Extended-Kalman-Filter) basiert auf der Verwendung des Kalman-Filters (Abschnitt 2.7.1), welcher mit Kenntnis des möglichen Messfehlers und der dazugehörigen fehlerhaften Beobachtung Rückschlüsse auf den aktuellen Zustand mit einer Fehlereliminierung ziehen kann. Der Algorithmus ist einer der ersten SLAM-Verfahren und verwendet die wahrscheinlichste Datenassoziation für seine Berechnungen.

Es ist ein online SLAM Algorithmus mit quadratischer update Komplexität, der zu jedem Zeitpunkt die aktuelle Roboterposition bereitstellt, welche allein für die weitere Abarbeitung verwendet wird.

Als Eingabeparameter für die Berechnung einer neuen Position werden die letzte bekannte Roboterposition, die Kovarianzmatrix, die Umgebungskarte, der Bewegungsvektor seit der letzten Position und die erkannten Landmarken verwendet. Aus diesen Daten generiert der EKF-SLAM

eine neue wahrscheinliche Roboterposition mit der dazugehörigen Kovarianzmatrix und der erweiterten Umgebungskarte.

Es wird eine Feature basierte Kartendarstellung verwendet, die auf punktuellen Landmarken basiert. Da der Berechnungsaufwand sehr groß ist, darf die Karte nicht zu viele Landmarken enthalten (i.d.R. weniger als 1000 [TFB06, S.312]). Außerdem muss eine Zweideutigkeit vermieden werden, um mögliche Fehler bei der Positionsschätzung zu minimieren. Aus diesem Grund wird ein zuverlässiger Feature Detektor vorausgesetzt. Der Algorithmus bietet die Möglichkeit Features mit und ohne Zuordnung zu verarbeiten, wobei es bei unbekannter Zuordnung wichtig ist, dass die Features nicht zu nah beieinander liegen.

Für die Roboter Positionsbestimmung wird ein Gaußsches Rauschen angenommen, weswegen die Unbestimmtheit gering gehalten werden muss, da sonst nicht tolerierbare Fehler entstehen, die durch die lineare Abarbeitung zu nicht ausgleichbaren Fehlern führen.

Graph SLAM löst das Full-SLAM Problem und ist ein offline Algorithmus, welcher wie der EKF-SLAM eine Feature basierte Karte aufbaut, da er erst, wenn alle Daten (die gesamte Bewegung) gesammelt wurden, ein endgültiges Ergebnis liefert. Während der Verarbeitung werden alle Positionen des Roboters und der Features in der Karte gespeichert und für jeden weiteren Bearbeitungsschritt weiterverwendet. Die dabei entstehende Karte ist ein spärlicher Graph (sparse graph), der sich über die Features und die Information darüber, aus welchen Positionen diese Features erkannt wurden, aufbaut. Das Erweitern des Graphen ist im Vergleich zum Erweitern der Kovarianzmatrix im EKF-SLAM deutlich einfacher zu berechnen.

Graph SLAM bildet seine Karten, indem er neue Informationen in den Graph einträgt, ohne sie vorher aufzulösen oder Berechnungen anzustellen. Somit kann er Karten bauen, die deutlich größer sind als im EKF-SLAM.

Da Graph SLAM alle Positionen speichert und erst hinterher das Kartieren durchführt, kann er im Nachhinein vorhandene Assoziationen und Berechnungen erneut überarbeiten und somit eine höhere Genauigkeit in der Karte erzielen, als das bei dem EKF-SLAM der Fall ist. Jedoch liegt ein Nachteil in der linearen Zunahme an benötigtem Speicher, da der Graph kontinuierlich aufgebaut wird und somit mit jedem Schritt wächst und erst bei der Berechnung nach der Datenakquirierung die Assoziationen bestimmt werden.

SEIF SLAM (Sparse Extended Information Filter) ist ein online SLAM Verfahren, welches wie der EKF-SLAM nur die aktuelle Roboterposition und die Karte speichert, um daraus die Folgeposition zu bestimmen. Er ist ebenso wie der EKF-SLAM und der Graph SLAM ein Algorithmus, der eine Feature basierte Karte aufbaut. Jedoch wird hierfür eine Datenrepräsentation wie in dem Graph SLAM verwendet. Damit wird in dem Algorithmus das Beste aus dem EKF-SLAM und dem Graph SLAM vereint, der online SLAM Teil des EKF-SLAM und die genauere und mehr wissende Karte des Graph SLAM. Damit benötigt SEIF SLAM weniger Rechenleistung als der Graph SLAM.

Im Gegensatz zu Graph SLAM-Verfahren ist es bei SEIF SLAM Algorithmen möglich Verknüpfungen zwischen Landmarken zu modellieren (z.B. auf Basis der örtlichen Nähe). Der Vorteil

dieser Erweiterung besteht darin, dass die aus verschiedenen Roboterpositionen erkannten Landmarken nicht nur als einzelne Marken verglichen werden müssen, sondern auch als Cluster betrachtet werden können. Somit können Korrespondenzen einfacher ermittelt werden. Für die Aktualisierung dieser Karte wird nur eine konstante Zeit benötigt, die unabhängig von der Größe der Karte ist, wodurch der Algorithmus viel effizienter als der EKF-SLAM ist.

Fast SLAM ist der einzige SLAM Algorithmus in [TFB06] der einen Partikelfilter verwendet und das Online und Full SLAM Problem löst.

Der Fast SLAM speichert im Vergleich zu den anderen SLAM-Verfahren nicht nur eine aktuelle Roboterposition, sondern eine vorher definierte Partikelanzahl (Positionen) und die dazugehörigen Umgebungskarten. Wenn eine neue Roboterbewegung erfasst wurde, bestimmt ein Bewegungsalgorithmus eine Position (siehe Abbildung 2.9) an der sich der Roboter jetzt befinden könnte. Zu dieser Position wird ein Wahrscheinlichkeitswert ermittelt und die dazu passende Karte aufgebaut.

Nachdem bei jedem Partikel diese Schätzung durchgeführt wurde, wird aufgrund der Wahrscheinlichkeiten mit dem Resampling entschieden, welche der neuen Positionen für die nächste Betrachtung verwendet und welche verworfen werden. Das Resultat ist eine Liste von möglichen Roboterpositionen, wobei die wahrscheinlichsten Positionen mehrfach vorkommen. Durch die Verwendung des Partikelfilters wird der SLAM Algorithmus robuster gegen fehlerhafte Feature Zuordnungen.

Der Fast SLAM Algorithmus ist der erste SLAM, der in dem Buch von Thrun et al. [TFB06] nicht nur Feature basiert ist, sondern die Möglichkeit bietet reine Laserscanner Daten zu verwenden. Hierfür wird statt einer Feature basierten Karte eine Belegungsraaster Karte (Occupancy Grid Map) verwendet. Die Wahrscheinlichkeit für eine Roboterposition wird berechnet, indem eine Rasterkarte anhand der Sensordaten und der Roboterposition erstellt wird und diese mit der Umgebungsrasterkarte verglichen wird. Anhand der Übereinstimmungen beider Karten wird eine Wahrscheinlichkeit für die jeweilige Roboterposition bestimmt. Diese Wahrscheinlichkeit wird dann für das Resampling verwendet.

Der Nachteil an dem Fast SLAM besteht darin, dass für jeden verwendeten Partikel eine extra Karte gespeichert wird. Dies ist unabhängig davon, ob eine Feature basierte Karte oder eine Rasterkarte verwendet wird.

DP-SLAM basiert auf Fast SLAM [TFB06] und löst das Online- und Full SLAM Problem [EP]. Da in den letzten Jahren die Laserscanner preisgünstiger geworden sind, konzentrieren sich die Entwickler auf die Verwendung des Fast SLAM ohne Features, wodurch eine Vorverarbeitung zur Feature Erkennung nicht nötig ist. Es besteht jedoch weiterhin das Problem der Speicherung, da der naive Partikelfilter SLAM Ansatz für jeden Partikel eine Karte im Speicher vorhält. Bei jeder Vervielfältigung von Partikeln während des Resamplings muss diese Karte kopiert werden, was zu erhöhten Lese- und Schreiboperationen führt. Außerdem wird mit fortschreitender Erkundung der Umgebung und sich vergrößernder Karte immer mehr Speicher benötigt.

Das Speicherplatzproblem wird mit der Einführung einer neuen Kartenrepräsentation versucht zu lösen, indem eine Karte mit verteilten Partikeln (distributed particle (DP) mapping) generiert

wird. Damit wird eine kürzere Laufzeit erreicht und der Algorithmus ist in der Lage Schleifen (loops) in über 60 Meter Entfernung ohne die Verwendung von externen Loop-closure Algorithmen zu schließen, da er exakte Karten erstellt.

Die Lösung für das Problem ist die Einführung eines Abstammungsbaumes für Partikel. Wenn aus einem Partikel im Iterationsschritt i ein Nachfolger Partikel $i+1$ erstellt wird, wird die Vorgänger- (Elternpartikel) bzw. Nachfolgerrelation (Kinderpartikel) explizit gespeichert. Werden aus einem Partikel mehrere Nachfolger erzeugt, sind diese Geschwister. Führt man nun einen Laserscann durch, der einen Bereich A in der Karte M mit $A \ll M$ erzeugt, so werden in dem Kind nur die neue Roboterposition und die Unterschiede zwischen A und M gespeichert. Somit speichert das Kind maximal A Karteninformationen.

Zur weiteren Optimierung werden Elternknoten, die nach der Nachfolgerzeugung keine Kinderknoten bekommen, aus dem Baum gelöscht. Außerdem werden Elternknoten, die nur einen Kindknoten besitzen, zusammengeführt um die Tiefe des Baumes zu minimieren.

Die Herausforderung dieser Kartenstruktur ist einen effizienten Aktualisierungs- und Lokalisierungsalgorithmus zu entwerfen. Hierfür wird der alte Ansatz, indem ein Partikel mit einer Karte in Verbindung gebracht wird, verworfen und es wird eine Karte gebildet, über die mehrere Partikel verteilt sind. Somit bestehen die Raster der Karte aus Bäumen, die über die IDs der Partikel aufgespannt werden und über eine Suche in dem jeweiligen Baum den Wert des aktuellen Rasters zurückliefern. Im Abschnitt 4.8 wird diese Kartenrepräsentation näher erläutert.

Dieser Ansatz verringert den benötigten Speicherplatz und erhöht den Rechenaufwand aufgrund der Suche in der Karte. Es wurde festgestellt, dass trotz des erhöhten Suchaufwandes ein deutlicher Geschwindigkeitszuwachs im Vergleich zum herkömmlichen Fast SLAM Algorithmus zu verzeichnen ist.

Core-SLAM oder auch Tiny-SLAM genannt wurde von Steux und Hamzaoui [SH] entwickelt und löst das Online SLAM Problem. Der Algorithmus ist eine Weiterentwicklung des Fast SLAM Algorithmus [TFB06] mit dem Ziel einzelne Räume zu kartieren. Eine Implementierung dieses Algorithmus ist auf der OpenSLAM Plattform [ope] veröffentlicht. Der Core SLAM Algorithmus benötigt weniger Speicherplatz, indem er im Gegensatz zum DP-SLAM nur eine Karte verwendet. Hierbei wird das Partikelverfahren insofern verwendet, dass mehrere Partikel zu einer Position bestimmt werden und geprüft wird, welcher der wahrscheinlichste Partikel ist. Mit diesem wird die neue Roboterposition festgelegt und die Karte aktualisiert.

Diese Erweiterung ist auf das Kartieren von einzelnen Räumen ausgelegt und schließt das Vorhandensein größerer Korridore aus. In Korridoren kann der Roboter im Vergleich zum DP-SLAM die Orientierung verlieren und seine Position so ungenau schätzen, dass der Fehler nicht mehr behoben werden kann. Da bei dem Core SLAM eine genaue Odometrie des Roboters vorausgesetzt wird, können kleine Schleifen problemlos geschlossen, wohingegen für größere Schleifen extra Loop Closure Algorithmen benötigt werden.

An diesen Beispielen kann man sehr gut sehen, wie der Fast SLAM Algorithmus verbessert und an andere Systemvoraussetzungen (Laser Scanner ohne Feature Punkte) angepasst wurde. Mit Hilfe des Core-SLAM wurde diese Verbesserung noch einmal weiterentwickelt und an einen spe-

ziellen Anwendungsfall mit Umgebungen, in denen keine oder kaum Korridore aber Einzelräume existieren, die effizient kartiert werden sollen, erweitert.

Ein weiterer Anwendungsfall ist in der Belegarbeit von Engelhardt [Eng] beschrieben. In der Arbeit geht es darum ein Online SLAM-Verfahren zu implementieren, das in Echtzeit ausgeführt werden kann und hierfür in regelmäßigen Abständen neue Sensordaten zur Verfügung gestellt bekommt. Der Autor betrachtet in diesem Zusammenhang mehrere SLAM Verfahren (SEIF-, Graph-, Fast- und Tiny SLAM) und wählt den Tiny SLAM aus, da dieser für seine Aufgabenstellung am geeignetsten ist. Tiny SLAM verwendet das Monte-Carlo Lokalisationsverfahren um die Roboterposition zu ermitteln, welches in diesem Algorithmus am rechenaufwendigsten ist. Nach der Positionsermittlung wird die Karte, welche aus Laserscanner Daten erstellt wird, anhand der bekannten Roboterposition um die neuen Umgebungsdaten erweitert.

Für die Echtzeitfähigkeit wurde der Algorithmus so modifiziert, dass er Sensordaten verwirft, die während einer aktuellen Berechnung eintreffen. Außerdem wird die Lokalisationsberechnung nach einer Zeitüberschreitung abgebrochen und das bis dahin beste Ergebnis verwendet.

Gmapping SLAM ist ein SLAM Verfahren, das das Online- und Full SLAM Problem löst. Der Algorithmus ist Laserscanner basiert und verwendet einen horizontalen Laserscann der Umgebung, mit welchem er eine occupancy Grid Map aufbaut. Der Algorithmus ist speziell für Indoor Umgebungen konzipiert und in die ROS Plattform [ros] als verfügbarer SLAM integriert. Die ROS Implementierung ist ebenfalls auf der OpenSLAM Plattform [ope] veröffentlicht.

Der Gmapping SLAM ist ein Rao Blackwellized Partikelfilter basiertes Verfahren, das für jeden Partikel eine eigene Grid Map im Speicher verwaltet. Die Berechnung der Partikel besteht darin, dass zu Beginn mithilfe der Odometriedaten des Roboters und eines Bewegungsmodells zu jedem Partikel eine wahrscheinliche Folgeposition berechnet wird. Diese Position wird mit einer Gaußkurve optimiert und damit eine Wahrscheinlichkeitsberechnung durchgeführt in der die Sensordaten mit der dazugehörigen Karte verglichen und ein Wahrscheinlichkeitswert für jeden Partikel bestimmt wird. Basierend auf diesem Wahrscheinlichkeitswert wird dann ein Resampling Verfahren, in diesem Fall Systematic Resampling, verwendet um die endgültige Partikelliste zu erhalten.

In dem Algorithmus besteht die Möglichkeit die Position des Roboters weiter zu verbessern, indem ein zusätzlicher Positionierungsschritt (ICP Algorithmus) für die Partikel durchgeführt wird. Der Vorteil ist, dass die Position sehr genau bestimmt werden kann, jedoch benötigt dieses Verfahren zusätzliche Rechenleistung, was die Onlinefähigkeit beeinflusst.

RBPF-SLAM ist ein Partikelfilter SLAM Ansatz, der von Schroeter und Gross [SG08] beschrieben wird. Er verwendet einen horizontalen Laserscann und erzeugt mithilfe dieser Daten eine occupancy Grid Map. Im Vergleich zu einfachen partikelbasierten Verfahren verwendet dieser SLAM ebenfalls einen Rao-Blackwellized Partikelfilter. Somit können genauere bzw. wahrscheinlichere Partikelpositionen erzeugt werden. Es besteht die Möglichkeit die Anzahl der verwendeten Partikel zu minimieren und trotzdem die gleiche Genauigkeit zu erzielen.

Dieses SLAM-Verfahren verwendet ebenso wie das Gmapping die Odometriedaten des Roboters um mithilfe eines Bewegungsmodells die neuen Roboterpositionen zu berechnen. Jedoch werden

diese Positionen mithilfe des Kalman Filters verbessert und zuletzt wird wieder die Wahrscheinlichkeit mit einem Algorithmus berechnet. In dem MRPT-Framework [mrp] in dem der RBPF-SLAM implementiert ist, gibt es für die Wahrscheinlichkeitsberechnung eine Auswahl zwischen 7 verschiedenen Algorithmen auf die im Abschnitt 4.4 eingegangen wird.

Zum Abschluss wird ein Resampling durchgeführt, wobei wiederum die Auswahl zwischen 4 verschiedenen Verfahren besteht, die im Abschnitt 4.7 erläutert werden.

ICP-SLAM wird von Segal et al. [SHT] erläutert und ist ein Online SLAM Verfahren, das wie der RBPF-SLAM im MRPT-Framework [mrp] implementiert, jedoch im Gegensatz zu dem RBPF kein partikelbasierter Ansatz ist. Dieser SLAM verwendet den ICP-Algorithmus zum Bestimmen seiner Position auf der Karte. Die Karte ist hier wie in den zuvor betrachteten SLAM-Verfahren eine occupancy Grid Map. Da es keine verschiedenen Partikel gibt, existiert nur genau eine Karte.

Die Vorgehensweise des ICP-SLAM ist konzeptionell folgendermaßen aufgebaut. Es besteht die Möglichkeit mit den Odometrieinformationen des Roboters eine Positionsschätzung über ein Bewegungsmodell durchzuführen, dies ist aber nicht unbedingt nötig. Jedoch kann die Positionsschätzung die Berechnungszeit des ICP beschleunigen und Fehler minimieren. An dieser Stelle kommt der ICP-Algorithmus zum Einsatz, der die Roboterposition auf der Karte mithilfe der Sensordaten, ohne Roboter Bewegungsdaten zu verwenden, ermittelt. Nach der Positionsermittlung muss noch die Karte aktualisiert werden. Für diesen Anwendungsfall stehen in dem MRPT-Framework 3 verschiedene ICP Varianten zur Auswahl.

Grid-SLAM ist eine SLAM Implementierung, die auf der OpenSLAM Plattform [ope] veröffentlicht wurde und das Online sowie Full SLAM Problem löst. In dieser SLAM Implementierung gibt es keine Neuerungen im Vergleich zu den zuvor erläuterten Verfahren. Grid-SLAM ist ein partikelbasiertes Verfahren ebenso wie Gmapping und speichert auch für jeden Partikel eine eigene Karte als occupancy Grid Map.

Das Vorgehen beider Algorithmen ist somit in ihren Berechnungen identisch, mit dem einzigen Unterschied, dass in der Grid-SLAM Implementierung kein ICP verfügbar ist. Die restlichen Berechnungsschritte sind im Vergleich beider Algorithmen nur in ihrer Implementierung (Rechenabfolgen) und der Programmstrukturierung (Klassen- und Methodenstrukturierung) unterschiedlich, was aber für das Ergebnis des Algorithmus irrelevant ist, da keine Hardware oder Speicheroptimierungen entdeckt wurden.

Hector-SLAM ist ein von Kohlbrecher et al. [KMG⁺] entwickeltes SLAM Verfahren, das in die ROS-Plattform [ros] integriert ist und das Online SLAM Problem löst. Der Algorithmus hat im Vergleich zu den zuvor genannten Verfahren einen neuen Aufbau und stellt für seinen Verwendungszweck (Such- und Rettungsaufgaben) spezielle neue Methoden, wie z.B. das Erkennen von Personen bereit. Jedoch wird auch hier als Kartenrepräsentation eine occupancy Grid Map verwendet und es gibt nur eine Karte, da es kein partikelbasierter Ansatz ist.

Als Sensordaten werden Daten aus einem Laserscanner verwendet oder Daten, die durch andere Sensorik gewonnen wurden. Da der Hector-SLAM für den Rettungseinsatz entwickelt wurde,

gibt es noch andere Sensordaten in Form von Features, wie z.B. QR-Codes die u.a. zu rettende Personen simulieren.

Das Lokalisieren des Roboters auf der Karte wird wie in anderen Algorithmen zuerst mithilfe der Odometriedaten des Roboters und einem Bewegungsmodell durchgeführt. Danach wird ein Kalman Filter angewendet, der mithilfe der Sensordaten die Wahrscheinlichkeitsberechnungen durchführt und die Roboterposition auf der Karte bestimmt. Somit wird die Positionsschätzung wie beim EKF-SLAM durchgeführt.

SLAM 6D ist ein Online SLAM-Verfahren, das auf der OpenSLAM Plattform [ope] veröffentlicht ist und der erste unter den bereits erwähnten Algorithmen, der 3-dimensionale Raumdaten verarbeiten kann. Er verwendet als Karte eine Octo Map, was einer 3-dimensionalen occupancy Grid Map entspricht, wovon nur eine im Speicher gehalten wird, da es kein partikelbasierter Ansatz ist. Als Sensor wird eine Tiefenbildkamera, wie z.B. die Microsoft Kinect verwendet.

Gleichwohl das Bewegungsmodell die Ergebnisqualität verbessert, kann der Algorithmus auch ohne dieses Modell eingesetzt werden. Die Positionsschätzung wird mithilfe eines ICP Algorithmus durchgeführt, jedoch wird im Vergleich zum ICP-SLAM ein ICP für 3-dimensionale Sensor- und Kartendaten benötigt. Da der Rechenaufwand beim Verwenden der kompletten Karte zu hoch ist, wird die Transformation jeweils nur zwischen zwei Sensor Aufnahmen berechnet. Aus diesem Grund sind die Karteninformationen zusätzlich noch als Graph mit den verschiedenen berechneten Roboterpositionen und den dazugehörigen Sensordaten gespeichert.

RGBD-SLAM oder Online 6D SLAM genannt, ist ebenso wie der SLAM 6D ein Online SLAM, der auf der OpenSLAM Plattform [ope] veröffentlicht ist. Weiterhin verwendet das Verfahren ebenso 3-dimensionale Sensordaten und Kartenrepräsentationen. Jedoch benötigt er im Gegensatz zu dem SLAM 6D nicht nur ein Tiefenbild, sondern auch das dazugehörige RGB-Bild und die Zuordnung der Pixel zu den Raumkoordinaten. Hierfür eignet sich die Kinect sehr gut, da sie RGBD-Bilder aufnimmt und als Ausgabe den Tiefenbildkoordinaten die passenden RGB Pixelpositionen zuordnet.

Diese Zuordnung des Tiefenbildes und des RGB-Bildes ist für die Positionsschätzung wichtig. Hierbei besteht die Auswahl aus zwei verschiedenen Verfahren, die einzeln oder auch kombiniert verwendet werden können. Dies wären der *3D ICP-Algorithmus*, wie schon im SLAM 6D verwendet, und der *SIFT Feature Detektor*. Mit dem SIFT Feature Detektor können in jeweils 2 RGB-Bildern Features erkannt und zugeordnet werden. Nach der Zuordnung der Features zweier Aufnahmen kann die Transformation der beiden Roboterpositionen zueinander berechnet werden. Hieraus lässt sich nun der Bewegungsgraph bestimmen und die verschiedenen Roboterpositionen ableiten.

3.2 PLATTFORMEN

Es gibt eine Vielzahl Robotik-Plattformen und Frameworks, wie z.B. Orca³, Carmen⁴, ROS [ros], MRPT [mrp] usw.. Jedoch beschäftigt sich der Großteil dieser Plattformen nicht mit SLAM-Verfahren, sondern hauptsächlich mit den Hardware nahen Bereichen, wie z.B. der Robotersteuerung und Sensordatenauslesung sowie Verarbeitung. Diese stellen aber keine Implementierung von SLAM als Platfordienst zum Kartieren und Lokalisieren zur Verfügung. Die drei große Plattformen (ROS [ros], MRPT [mrp] und OpenSLAM [ope]) verwenden SLAM-Verfahren zum Kartieren und Lokalisieren und werden im Folgenden näher erläutert.

Open SLAM Plattform [ope] verfolgt das Ziel eine Vielzahl von SLAM Algorithmen und Implementierungen zu sammeln und diese einer breiten Masse für die Forschung und Anwendung bereitzustellen. Hierbei schreiben die Betreiber keine Programmiersprache vor, es ist nur wichtig, dass die Algorithmen stabil laufen. Ein überwiegender Teil der aktuell verfügbaren Implementierungen beschränkt sich auf die Programmiersprachen *Matlab* und *c/c++*.

Der Programmierer muss nur eine kurze Dokumentation schreiben in der er seinen SLAM vorstellt und die Systemvoraussetzungen (Hardware, Software, Sensordaten, Kartenstruktur etc.) angibt.

Aufgrund der großen Auswahl an SLAM-Verfahren bietet die Plattform eine gute Möglichkeit für Recherchen und gibt den Benutzern, die nur einen funktionierenden SLAM benötigen, eine große Auswahl und einfache Verwendungsmöglichkeit der Algorithmen. Möchte der Benutzer jedoch Funktionen der verschiedenen SLAM-Verfahren kombinieren und somit einen für seinen Anwendungsfall bzw. Hardwarevoraussetzungen verwendbaren SLAM bauen, so ist dies auf der OpenSLAM Plattform problematisch und schwer umzusetzen. Da es sich jeweils um eigenständige Implementierungen handelt, die in der Granularität von der kompletten Implementierung in einer Methode bis zu einer Aufspaltung in mehrere Methoden veröffentlicht sind. Darüber hinaus ist der Grad der Wiederverwendung gleicher Teile sehr gering, so dass häufig komplette Verfahren neu implementiert werden.

MRPT Plattform [mrp] geht im Vergleich zur OpenSLAM Plattform [ope] einen Schritt weiter und bietet ein komplettes Framework an, welches in *c/c++* programmiert ist. Das Framework schreibt Datenaustauschformate sowie Datenstrukturen vor, an denen sich die Entwickler orientieren müssen. Das Framework setzt sich aus drei Teilen zusammen:

Teil 1 ist die Bewegungsplanung mit verschiedenen Navigations- und Steuerungsalgorithmen.

Teil 2 ist die Computer Vision, welche verschiedene Algorithmen für die Analyse von Sensordaten bereitstellt. Dies wären z.B. Feature Erkennungsalgorithmen (SIFT, SURF, ...) und Feature Tracking Algorithmen, welche für einige SLAM-Verfahren zur Ermittlung von Features benötigt werden.

³<http://orca-robotics.sourceforge.net/> vom 24.06.2014

⁴<http://carmen.sourceforge.net/> vom 24.06.2014

Teil 3 beinhaltet die verschiedenen SLAM-Verfahren und die in diesem Zusammenhang verwendeten Algorithmen. Aktuell sind Graph-, EKF-, Fast-, RBPF- und ICP-SLAM Implementierungen und einige Algorithmen wie ICP und Monte-Carlo Lokalisierung verfügbar.

MRPT ist somit eine Plattform, die es dem Programmierer ermöglicht seinen SLAM Algorithmus im Ganzen selbst zusammenzustellen. Wie am Anfang von Abschnitt 3.1 angesprochen, hat jeder SLAM seine Voraussetzungen (z.B. Hardwareanforderungen, Kartenrepräsentation, verfügbare Sensorik, ...), die erfüllt sein müssen, damit er angewendet werden kann.

Betrachtet man dies am Beispiel des EKF-SLAM, sieht man dass dieser einen vorgelagerten Feature Erkennungsalgorithmus wie SIFT oder Harris benötigt, welche das Framework bereitstellt. Nach der Erkennung kann der Programmierer entweder den EKF-SLAM mit einer unbekanntenen Zuordnung verwenden oder einen weiteren Algorithmus zur Feature Zuordnung einfügen und dann den EKF mit bekannter Zuordnung anwenden.

Zu beachten ist, dass die MRPT Plattform diese einzelnen Funktionen (z.B. Feature Detektor, Kalman Filter, Bewegungsmodelle, ...) und Datenrepräsentationen (z.B. Grid Map, Octo Map, ...) zwar bereitstellt, jedoch keine Regeln definiert in welcher Art und Weise diese in einem SLAM-Verfahren verwendet werden müssen. Der Programmierer des SLAM-Verfahrens ist dafür zuständig, die verfügbaren Mittel korrekt und zielführend zu verwenden, um einen funktionierenden SLAM-Algorithmus zu bauen. Was auf dieser Plattform trotzdem noch zu umfangreichem Programmieraufwand führt, wie an den implementierten SLAM-Verfahren zu erkennen ist.

ROS Plattform ist ein OpenSource Projekt, das sich speziell mit Roboterumgebungen beschäftigt. Es stellt eine komplette Umgebung mit Schnittstellen und Implementierungen für Sensoren, Robotersteuerungen, Lokalisierung, Kartierung und Steuerung bereit. Im ROS werden auch SLAM-Verfahren angeboten, jedoch wird hier der SLAM Algorithmus noch mehr gekapselt und als komplette Einheit bezeichnet. Der SLAM Algorithmus bekommt die Möglichkeit alle Bewegungs- und Sensordaten abzurufen und gibt dann eine Karte zurück. Daraus kann man ableiten, dass die Wiederverwendung und Flexibilität stark einschränkt und dass der Wechsel zu einem anderen SLAM-Verfahren eine komplette Neuimplementierung zur Folge hat.

Neben der ROS Funktionalität gibt es noch die Möglichkeit über Log Files andere Frameworks wie OpenCV oder PointCloudLibrary einzubinden. Diese sind für die Verarbeitung von Sensordaten wie Kamerabilder oder Laserscanner Daten sehr interessant und hilfreich.

Auf der ROS Plattform sind die zwei SLAM-Verfahren GMapping [Abb06] und HectorSLAM [KMG⁺] implementiert.

3.3 ZUSAMMENFASSUNG

In diesem Abschnitt wurde ein Überblick über verschiedene SLAM-Verfahren gegeben. An den Beschreibungen ist zu erkennen, dass in vielen Algorithmen gleiche Funktionen verwendet bzw. diese nur erweitert wurden, wie z.B. beim GMapping, das im Vergleich zum Grid-SLAM zusätzlich einen ICP Algorithmus anbietet.

An diesen beschriebenen SLAM-Verfahren ist gut zu erkennen, dass die Verfahren aktuell nur einzeln betrachtet werden und in jeder Implementierung bzw. in jedem Anwendungsfall der Programmierer sich für den seiner Meinung nach am besten geeigneten Algorithmus entscheidet. In der Bachelorarbeit von Engelhardt [Eng] zur Umsetzung eines SLAM Algorithmus mit Echtzeitverhalten entscheidet sich der Autor für den TinySLAM, weil er sehr effizient und nicht zu rechenaufwendig ist. Die resultierenden negativen Fakten, wie eine unbestimmte Laufzeit, werden toleriert. Hierbei ist wieder gut zu erkennen, dass je nach spezieller Anwendung und Systemvoraussetzungen ein SLAM-Verfahren gewählt wird, sogar wenn es nicht vollständig den Ansprüchen entspricht.

Außerdem wurde ein Überblick über die verfügbaren Plattformen bzw. Frameworks gegeben, die SLAM Implementierungen bereitstellen bzw. diese zum Kartieren und Lokalisieren verwenden. Aus diesen Plattformen stammen die SLAM Algorithmen, die zum Analysieren und Extrahieren der Komponenten verwendet wurden.

Im Abschnitt 4 findet die Unterteilung der SLAM-Verfahren in die bereits erkennbaren Komponenten (Sensordaten, Bewegungsdaten, Partikelfilter, ...) statt und die Eigenschaften bzw. Funktionsweisen dieser werden beschrieben. Ziel ist es einen Baukasten zu erstellen, mit dessen Hilfe SLAM-Verfahren für spezielle Anwendungsfälle, wie z.B. Geschwindigkeit, Umgebungen usw., zusammengesetzt und generiert werden können.

4 KOMPONENTEN EXTRAKTION UND EIGENSCHAFTEN

An den zuvor beschriebenen SLAM-Verfahren ist eine Unterteilung in verschiedene Komponenten, wie z.B. Berechnung der Roboterposition, Partikelfilter, Resampling, Kartenaktualisierung, Kalman-Filter usw., erkennbar. Im Abschnitt 2 wurden grundlegende Eigenschaften und Funktionsweisen einiger Begriffe bereits erklärt, jedoch nicht auf ihre Integration und Funktion in SLAM-Verfahren eingegangen. Das wird in diesem Abschnitt nachgeholt, indem die Komponenten näher erklärt und zusätzlich ihre Funktion in SLAM-Verfahren betrachtet werden. Außerdem werden Beispielimplementierungen aus veröffentlichten Algorithmen bzw. einzelne Verfahren in Form von Pseudocode oder Gleichungen benannt und beschrieben. Hierbei wird auf benötigte Datentypen und Austauschformate eingegangen, um eine Alleinstellung der Komponenten und daraus resultierende Austauschmöglichkeiten der Implementierungen zu ermöglichen. Das Ziel ist es am Ende dieses Abschnittes einen Überblick über die Komponenten zu haben, aus denen SLAM-Verfahren zusammengesetzt werden können.

Das Hauptaugenmerk liegt in dieser Arbeit erst einmal nur auf Grid Map basierte Ansätze, die nicht unbedingt einen Feature Detektor benötigen. Wie aber zum Beispiel in der Erklärung des RGBD-SLAM schon erkennbar ist, findet hier zusätzlich zu der Positionsschätzung mit Hilfe des ICP Algorithmus die Verwendung von SIFT zur Verbesserung der Genauigkeit statt. Deswegen werden im Folgenden einige featurebasierte Ansätze angesprochen, jedoch in Kombination mit Grid Map basierten SLAM-Verfahren und nicht als eigenständige Feature basierte SLAM-Verfahren.

Begonnen wird mit der Benennung der identifizierten Komponenten in chronologischer Reihenfolge.

Wenn die Prozesse der bereits angesprochenen SLAM Algorithmen grob betrachtet werden, ergibt sich die Struktur aus Abbildung 4.1, in der zu erkennen ist, dass es mit dem Einlesen der Sensordaten und den dazugehörigen Bewegungsdaten des Roboters, falls vorhanden, beginnt. Diese Daten beschreiben die Bewegung des Roboters von der letzten Roboterposition aus, an welcher Sensordaten aufgenommen und verarbeitet wurden. Nachdem diese Daten akquiriert wur-

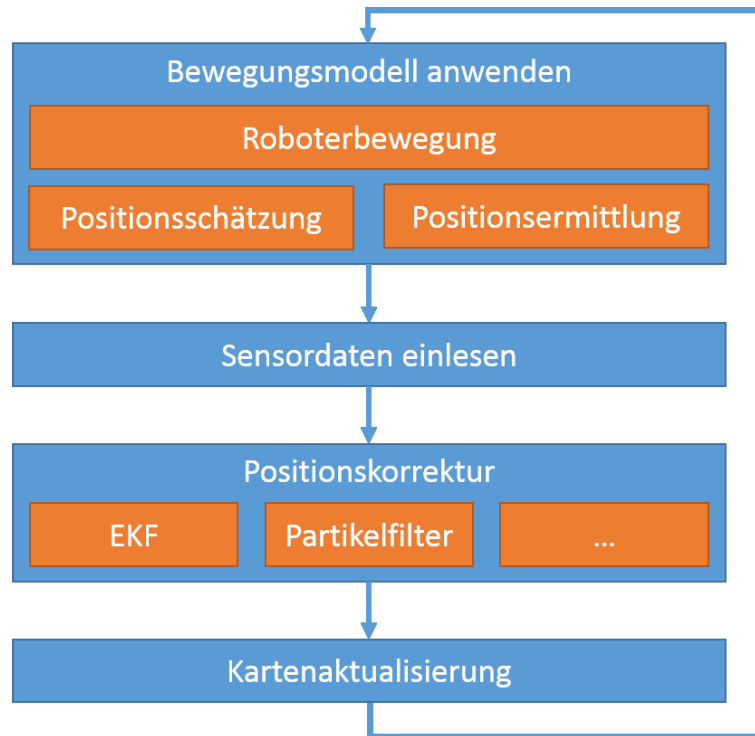


Abbildung 4.1: Modell der allgemeinen Prozessstruktur der Raster basierten SLAM-Verfahren

den, wird ein Bewegungsmodell angewendet, welches die wahrscheinlich nächste Position bzw. einen Bereich bestimmt, an dem sich der Roboter nun befindet. In Folge dessen wird die Positionsschätzung bzw. Verbesserung der Roboterposition mit Partikelfiltern oder Extended Kalman Filtern durchgeführt, was auch die Wahrscheinlichkeitsberechnung für die gefundene Position beinhaltet. Nachdem die finale Position des Roboters ermittelt wurde, werden die Sensordaten der Karte hinzugefügt.

4.1 SENSORDATEN AUFNEHMEN

Während der Recherchen ergaben sich verschiedene Sensortypen abhängig von den verschiedenen SLAM-Verfahren.

RGB-Kameras sind eine preisgünstige Variante, da sie in guter Qualität für unter 100Euro erhältlich sind. Hierfür werden jedoch Feature Detektoren benötigt, um verschiedene Aufnahmen ineinander zu transformieren. Außerdem stellen sie keine Tiefeninformationen bereit, weshalb diese aus mindestens zwei Aufnahmen von verschiedenen Roboterpositionen und den Bewegungsinformationen des Roboters extra berechnet werden müssen. Aufgrund der fehlerbehafteten Bewegungsinformationen und der Fehleranfälligkeit der Featuredetektoren sowie der Featurezuordnung sind die Tiefeninformationen ungenau und fehleranfällig.

Aus diesen Aufnahmen können Sensorinformationen in Form von Farbbildern oder Graustufenbildern erzeugt werden.

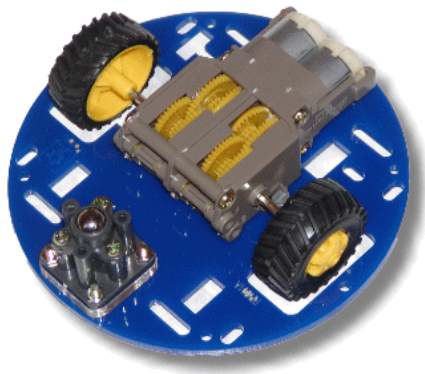


Abbildung 4.2: Roboter mit zwei unabhängig angetriebenen Rädern¹

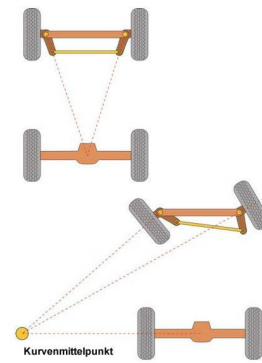


Abbildung 4.3: Roboter mit einer Antriebsachse und einer Lenkachse wie bei einem Kraftfahrzeug²

Laserscanner gibt es schon sehr lange auf dem Markt. Sie wurden zu Beginn zum Erzeugen von Grid Maps verwendet und können abhängig vom Laserscanner Modell 2-dimensionale Raumausschnitte in Form einer horizontalen Linie, meist in einem festen Öffnungswinkel von 180° bis 360°, oder 3-dimensionale Raumaufnahmen erzeugen. Wobei 2-dimensionale Laserscanner hauptsächlich verwendet werden. Wenn der Roboter sich im Innenbereich auf ebenen Flächen bewegt und der Laserscanner korrekt horizontal ausgerichtet ist, können hiermit gute Querschnitte von Räumen erzeugt werden. Ihr Vorteil ist im Vergleich zu RGB-Aufnahmen, dass sie direkt sehr genaue Tiefeninformationen liefern und aufgrund ihres großen Öffnungswinkels einen großen Raumausschnitt aufnehmen.

Tiefenbildkameras und RGBD-Kameras sind seit der Einführung der Kinect von Microsoft im Aufschwung und werden in vielen neuen Projekten verwendet. Dies ist auch im Bereich der SLAM-Verfahren der Fall. Hier besteht die Möglichkeit die 2-dimensionalen Anwendungen mit Laserscannern um die dritte Dimension zu erweitern und somit bessere Karten aufzubauen und die Aufnahmen lageunabhängiger zu machen. Es ist dabei nicht mehr so wichtig, dass der Sensor perfekt horizontal ausgerichtet ist. Außerdem sind die Tiefenbildkameras mit einem Preis von weniger als 1000 Euro eine gute Wahl. Die Kinect ist ein Beispiel für eine Kamera, die nicht nur ein Tiefenbild sondern auch noch das dazugehörige Farbbild mit einer Verknüpfung zum Tiefenbild liefert. Durch diese Kombination als RGBD-Bild können mit diesem Sensor die Herangehensweisen verschiedener SLAM-Verfahren kombiniert und somit u.a. die Genauigkeit gesteigert werden. Ein solches Beispiel ist der RGBD-SLAM.

4.2 BEWEGUNGSDATEN (ODOMETRIEDATEN) AUFNEHMEN

Abhängig von den verschiedenen Roboterarten können unterschiedliche Bewegungsdatenformate existieren. Diese Daten unterscheiden sich bereits in der Antriebsart. Eine weitverbreitete Variante ist die Verwendung von zwei Antriebsrädern (siehe Abbildung 4.2), die auf einer Achse

¹<http://www.rn-wissen.de/index.php/Roboter-Typen> vom 04.07.2014

²<http://de.wikipedia.org/wiki/Lenkung> vom 04.07.2014

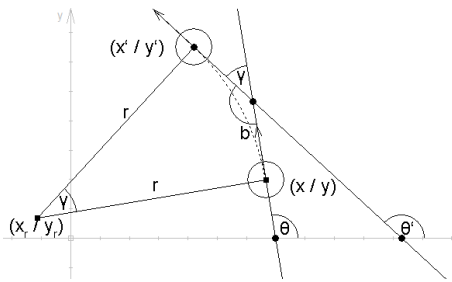


Abbildung 4.4: Velocity Modell Skizze der Bewegung des Roboters

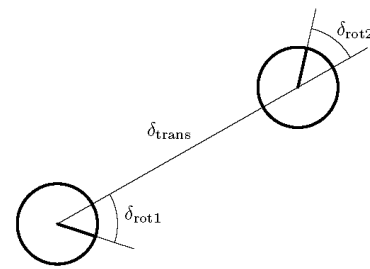


Abbildung 4.5: Odometrie Modell Skizze der Bewegung des Roboters [TFB06]

liegen, jedoch getrennt voneinander angetrieben werden. Jedes dieser Räder gibt über einen Inkrementalgeber die zurückgelegten Schritte aus. Dabei ist das Rad bei einer Sensorvariante z.B. in 12 gleichgroße Kreisabschnitte unterteilt, wobei ein Sensor die ihn passierten Abschnitte zählt und über diesen Wert die zurückgelegte Radstrecke mithilfe des Raddurchmessers berechnet. Mit den Systemparametern Radabstand, Raddurchmesser und Drehwinkel je Inkrementalgeber Schritt kann die Bewegung des Roboters berechnet werden. Dies wird z.B. in der Tiny-SLAM Implementierung durchgeführt. Eine weitere Möglichkeit wäre die Verwendung einer Antriebsachse und einer Lenkachse (siehe Abbildung 4.3), wie es von Kraftfahrzeugen bekannt ist. Hierfür werden jedoch andere Berechnungen als in der vorherigen Variante benötigt.

Es gibt darüber hinaus noch weitere Möglichkeiten einen Roboter anzutreiben, jedoch ist das für den SLAM-Algorithmus nicht von Interesse, da die Implementierungen meist nicht mit diesen proprietären Daten arbeiten. Die SLAM-Verfahren arbeiten hauptsächlich mit der Roboterposition (x, y, ϕ) (auch als Pose bezeichnet), welche sich im 2-dimensionalen Raum aus der Position auf der Karte (x, y) und der Blickrichtung ϕ des Roboters zusammensetzt. In dieser Arbeit werden folgende zwei Varianten für Bewegungsdaten unterschieden:

Velocity Modell mit dem Bewegungsvektor u_t , der aus einer Translationsgeschwindigkeit ν_t und einer Rotationsgeschwindigkeit ω_t zur Zeit t besteht. In der Abbildung 4.4 ist die Bewegung des Roboters von Position $(x; y)$ mit der Blickrichtung ϕ zu der Position $(x'; y')$ mit der Blickrichtung ϕ' dargestellt. Dabei bewegt sich der Roboter um den Kreismittelpunkt $(x_r; y_r)$ mit dem Radius r um den Winkel γ .

Aus der Abbildung 4.4 kann der Bewegungsvektor

$$u_t = \begin{pmatrix} \nu_t \\ \omega_t \end{pmatrix}$$

folgendermaßen abgelesen werden. Die Translation ν_t ist der Kreisbogen b in der Längeneinheit der Kartendarstellung. Die Rotationsgeschwindigkeit ω_t wird durch den Winkel γ im Bogenmaß (Rad) beschrieben, welcher gleich der Differenz der Roboterblickrichtungen $\phi' - \phi$ ist. Der Radius r lässt sich folgendermaßen berechnen:

$$r = \left| \frac{\nu_t}{\omega_t} \right|$$

Odometrie Modell (siehe Abbildung 4.5) besteht aus einer Anfangsrotation δ_{rot1} , einer Translationsbewegung δ_{trans} und einer Endrotation δ_{rot2} , die alle hintereinander ausgeführt, die Bewegung des Roboters beschreiben $(\delta_{rot1} \ \delta_{trans} \ \delta_{rot2})^T$. Der Bewegungsvektor ist in diesem Modell als Vektor u_t mit den Positionen des Roboters zum Startzeitpunkt \bar{x}_{t-1} und zum Endzeitpunkt \bar{x}_t definiert

$$u_t = \begin{pmatrix} \bar{x}_{t-1} \\ \bar{x}_t \end{pmatrix}$$

Mit diesen beiden Modellen können alle Roboterbewegungen im 2-dimensionalen Raum beschrieben werden. Somit muss das Modul, welches die Bewegungsdaten des Roboters verarbeitet, speziell auf den Roboter und seine Sensordaten angepasst sein und mithilfe dieser Sensordaten eine der zwei zuvor genannten Bewegungsrepräsentationen bestimmen und ausgeben. Mit diesen Daten kann das System dann weiterarbeiten.

Modelllose Variante ist eine Möglichkeit einen Fehler zu erzeugen, obwohl keines der beiden zuvor genannten Modelle verwendet wird. Dieser Fall ist im GMapping [ope] implementiert und stellt eine Bewegung durch den Bewegungsvektor u_t dar, der durch

$$u_t = \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta \phi \end{pmatrix}$$

beschrieben wird.

Die beschriebenen Algorithmen arbeiten alle im 2-dimensionalen Raum, wobei ebenfalls Umsetzungen für den 3-dimensionalen Raum bestehen, die in dieser Arbeit jedoch nicht berücksichtigt werden.

4.3 BEWEGUNGSMODELL (MOTION MODEL)

Die zwei verschiedenen Bewegungsdaten des Roboters werden benötigt, um die Roboterposition bzw. Positionsschätzungen abhängig vom verwendeten SLAM-Verfahren zu bestimmen. Hierbei kann zwischen *exakter Berechnung der Position* und *Positionsberechnung für Partikel* unterschieden werden. Diese werden im Folgenden näher erläutert.

4.3.1 Exakte Berechnung der Position

Für die exakte Berechnung der Position werden die Bewegungsdaten des Roboters verwendet, hierbei werden jegliche Fehlerparameter des Roboters ignoriert. Die Berechnung der Position x_t wird aus der vorherigen Position x_{t-1} und den Bewegungsdaten des Roboters zwischen den beiden Positionen ermittelt.

Velocity Model:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{\omega} \sin \theta + \frac{v}{\omega} \sin (\theta + \omega \Delta t) \\ \frac{v}{\omega} \cos \theta - \frac{v}{\omega} \cos (\theta + \omega \Delta t) \\ \omega \Delta t \end{pmatrix}$$

Odometrie Model:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} \delta_{trans} \cos (\theta + \delta_{rot1}) \\ \delta_{trans} \sin (\theta + \delta_{rot1}) \\ \delta_{rot1} + \delta_{rot2} \end{pmatrix}$$

Dies ist sinnvoll, wenn ein spezieller Algorithmus (z.B. ICP) für die Positionskorrektur verwendet wird, da in diesem Fall eine Ausgangsposition für die Berechnung hilfreich ist. Im konkreten Fall des ICP-Algorithmus bewirkt die Positionsberechnung, dass der Algorithmus die sukzessive Positionsannäherung mit weniger Berechnungsschritten abschließen kann und somit Rechenzeit einspart. Außerdem wird damit die Möglichkeit der Annäherung an falsche Positionen vermindert.

4.3.2 Positionsberechnung für Partikel

Die Positionsberechnung für Partikel unterscheidet sich von exakten Berechnung darin, dass der Algorithmus Informationen über den Roboterbewegungsfehler besitzt. Diese Fehlerwerte geben an, wie stark der Roboter bei den jeweiligen Bewegungen von dem Sollweg abweichen kann. Mit einem Zufallsgenerator und diesen Fehlerinformationen werden exakte Positionen im Fehlerraum berechnet und ausgegeben.

Die Partikelpositionen können auf drei verschiedene Arten berechnet werden.

I. Bewegungsfehler im Velocity Modell sind über die Parameter $\alpha_1, \alpha_2, \dots \alpha_6$ folgendermaßen definiert:

Fehler für v :

$$\alpha_1 \hat{=} \frac{\text{Abweichung (in m)}}{\text{zurückgelegtem Weg (in m)}}$$

$$\alpha_2 \hat{=} \frac{\text{Abweichung (in m)}}{\text{gedrehtem Winkel (in rad)}}$$

Fehler für ω :

$$\alpha_3 \hat{=} \frac{\text{Abweichung (in rad)}}{\text{zurückgelegtem Weg (in m)}}$$

$$\alpha_4 \hat{=} \frac{\text{Abweichung (in rad)}}{\text{gedrehtem Winkel (in rad)}}$$

Fehler für Winkel ϕ des Roboters:

$$\alpha_5 \triangleq \frac{\text{Abweichung (in rad)}}{\text{zurückgelegtem Weg (in m)}}$$

$$\alpha_6 \triangleq \frac{\text{Abweichung (in rad)}}{\text{gedrehtem Winkel (in rad)}}$$

Die Wahrscheinlichkeiten und Beispielpositionen für die Roboterbewegung können mit einer *Normalverteilung* über eine Gaußkurve bzw. eine *Dreiecksverteilung* bestimmt werden. Ein Beispiel hierfür ist aus [TFB06] die Positionsbestimmung für die Partikelpositionen mit einer Fehlerfunktion:

$$\nu_{error} = \nu + \text{sample}(\alpha_1 * \nu^2 + \alpha_2 * \omega^2)$$

$$\omega_{error} = \omega + \text{sample}(\alpha_3 * \nu^2 + \alpha_4 * \omega^2)$$

$$\phi_{error} = \phi + \text{sample}(\alpha_5 * \nu^2 + \alpha_6 * \omega^2)$$

Die folgenden Algorithmen zur *Normalverteilung* und *Dreiecksverteilung* berechnen nach Thrun et al. [TFB06] mit einem Zufallsgenerator (*rand()*) eine Beispielposition auf einer Gaußkurve bzw. Dreieckskurve mit der maximalen Abweichung von b^2 .

I. Normalverteilung:

$$\text{return } \frac{1}{2} \sum_{i=1}^{12} \text{rand}(-b, b)$$

II. Dreiecksverteilung:

$$\text{return } \frac{\sqrt{6}}{2} [\text{rand}(-b, b) + \text{rand}(-b, b)]$$

(*rand(x,y)* berechnet einen Zufallswert im Bereich $[x, y]$)

II. Bewegungsfehler im Odometrie Modell sind über Parameter $\alpha_1, \alpha_2, \dots, \alpha_4$ folgendermaßen definiert:

Fehler für die Rotationen δ_{rot1} und δ_{rot2} :

$$\alpha_1 \triangleq \frac{\text{Abweichung (in rad)}}{\text{gedrehtem Winkel (in rad)}}$$

$$\alpha_2 \triangleq \frac{\text{Abweichung (in rad)}}{\text{zurückgelegtem Weg (in m)}}$$

Fehler für δ_{trans} :

$$\alpha_3 \hat{=} \frac{\text{Abweichung (in m)}}{\text{zurückgelegtem Weg (in m)}}$$

$$\alpha_4 \hat{=} \frac{\text{Abweichung (in m)}}{\text{gedrehtem Winkel (in rad)}}$$

Die Wahrscheinlichkeiten und Beispielpositionen für die Roboterbewegung können mit einer Normalverteilung über eine Gaußkurve bzw. eine Dreiecksverteilung bestimmt werden. Ein Beispiel hierfür ist aus [TFB06] die Positionsbestimmung für die Partikelpositionen mit einer Fehlerfunktion (Sample ist wie im Velocity Modell definiert):

$$\delta_{rot1_{error}} = \delta_{rot1} + \text{sample}(\alpha_1 * \delta_{rot1}^2 + \alpha_2 * \delta_{trans}^2)$$

$$\delta_{trans_{error}} = \delta_{trans} + \text{sample}(\alpha_3 * \delta_{trans}^2 + \alpha_4 * \delta_{rot1}^2 + \alpha_4 * \delta_{rot2}^2)$$

$$\delta_{rot2_{error}} = \delta_{rot2} + \text{sample}(\alpha_1 * \delta_{rot2}^2 + \alpha_2 * \delta_{trans}^2)$$

III. Modelllose Variante wird über die Parameter $\alpha_1, \alpha_2, \dots, \alpha_4$ folgendermaßen definiert:

$$\alpha_1 \hat{=} \frac{\text{Abweichung (in m)}}{\text{zurückgelegtem Weg (in m)}}$$

$$\alpha_2 \hat{=} \frac{\text{Abweichung (in m)}}{\text{gedrehtem Winkel (in rad)}}$$

$$\alpha_3 \hat{=} \frac{\text{Abweichung (in rad)}}{\text{zurückgelegtem Weg (in m)}}$$

$$\alpha_4 \hat{=} \frac{\text{Abweichung (in rad)}}{\text{gedrehtem Winkel (in rad)}}$$

$$\alpha_5 \hat{=} 0.3 \alpha_1$$

Die Partikelposition über die Fehlerfunktion ergibt sich mit dem Sample aus den vorherigen Modellen mit:

$$\Delta x_{error} = \Delta x + \text{sample}(\alpha_1 * \Delta x + \alpha_2 * \Delta \phi + \alpha_5 \Delta y)$$

$$\Delta y_{error} = \Delta y + \text{sample}(\alpha_1 * \Delta y + \alpha_2 * \Delta \phi + \alpha_5 \Delta x)$$

$$\Delta \phi_{error} = \Delta \phi + \text{sample}(\alpha_4 * \Delta \phi + \alpha_3 * \sqrt{\Delta x^2 + \Delta y^2})$$

4.4 WAHRSCHEINLICHKEITSBERECHNUNG (LIKELIHOOD MODEL)

Die Wahrscheinlichkeitsberechnungsfunktion ermittelt zu einer Karte, Roboterposition und den dazugehörigen Sensordaten einen Wert, der ausdrückt, wie wahrscheinlich es ist, dass diese Roboterposition mit der realen Position im Raum übereinstimmt. Hierbei ist zu beachten, dass der Wahrscheinlichkeitswert kein Wert zwischen 0 und 1 sein muss. Er ist meist in Relation zu anderen ermittelten Werten zu betrachten, womit die Algorithmen für Partikelfilter SLAM-Verfahren benötigt werden.

Als Grundlage dienen für diese Berechnungen Laserscanner oder Sonarsensor Daten, die in Bezug zu Grid Maps (Occupancy Grid Map oder Octree Map) betrachtet werden.

Im Folgenden werden 2D basierte Algorithmen näher erklären, die im Zusammenhang mit Grid Maps und horizontalen Linien Laserscannern arbeiten. Tiefenbildkameras können auch verwendet werden, jedoch ist eine vorherige Extraktion einer Laserlinie nötig.

Übereinstimmungswahrscheinlichkeit (Likelihood Consensus) bezeichnet eine Methode aus dem MRPT Framework. In dem Algorithmus wird eine Karte anhand der Sensorinformationen im globalen Koordinatensystem der Umgebungskarte aufgebaut. Alle belegten Zellen in der Sensorkarte werden mit den dazugehörigen Zellen in der Umgebungskarte verglichen und die Ergebnisse aufsummiert. Der Pseudocode in Listing 4.1 beschreibt das Verfahren.

Listing 4.1: Berechnung der Übereinstimmungswahrscheinlichkeit von horizontalen Laserscannern.

```
Sensorkarte = Karte der Sensordaten mit
                Roboterposition als Referenz;
likelihood = 0;
counter = 0;
foreach (belegte Zellen in der Sensorkarte)
{
    likelihood += 1 - Umgebungskarten Wert an
                aktueller Position in Sensorkarte;
    counter++;
}
if (counter > 0)
{
    likelihood /= counter;
}
sigma = Sigma Parameter des Laserscanners;
likelihood = likelihood hoch sigma;
return log(likelihood);
```

Die Methode Zellendifferenz ist im MRPT implementiert. Der Algorithmus erstellt anhand der Sensordaten im globalen Koordinatensystem der Umgebungskarte eine eigene Karte. Darin prüft er die Übereinstimmung in allen freien und belegten Zellen, wie im Algorithmus 4.2 beschrieben.

Die Methode Raytracing ist im MRPT implementiert. Der Algorithmus verwendet die Laserlinien und berechnet zu jeder eine Linie auf der Umgebungskarte. Die Wahrscheinlichkeit des Laserscans wird rechnerisch mithilfe der Länge der Sensor Laserlinien und ihrer berechneten Linien aus der Umgebungskarte ermittelt. Im Listing 4.3 ist das Verfahren beschrieben.

Neben den vorgestellten Algorithmen gibt es noch zahlreiche weitere, die hier jedoch nicht erläutert werden. Wie an den Beispielen zu erkennen ist, bekommen die Algorithmen als Eingabe die Sensordaten und eine Roboterposition mit denen dann die Wahrscheinlichkeit berechnet und als reelle Zahl zurückgegeben wird.

Listing 4.2: Berechnung der Wahrscheinlichkeit anhand eines Kartenvergleichs.

```

Sensorkarte = Karte der Sensordaten mit Roboterposition als Referenz;
differenceCounter = 0;
counter = 0;
foreach (freie und belegte Zellen in der Sensorkarte)
{
    c1 = Zelle in der Sensorkarte;
    c2 = Zelle in der Umgebungskarte an gleicher Position wie c1;
    if (c1 == frei && c2 == belegt ||
        c1 == belegt && c2 == frei) {
        differenceCounter++;
    }
    counter++;
}
if (counter > 0) {
    likelihood /= counter;
}
likelihood = 1 - differenceCounter / counter;
return log(likelihood);

```

Listing 4.3: Berechnung der Wahrscheinlichkeit mithilfe von Raytracing.

```

lOriginal = Laenge der Laserlinie vom Sensor;
lKarte = Laenge der Laserlinie in der Umgebungskarte;
ergebnis = 1;
foreach (Fehlerfreien Laserlinien des Sensors) {
    likelihood = 0.1 / sensorMaxRange + 0.9 * pow(e,
        (-1*min(lKarte-lOriginal, 2.0) / Laserscann Sigma*sqrt(2)) );
    ergebnis += log(likelihood);
}
return ergebnis;

```

4.5 KALMAN FILTER

Der Kalman Filter setzt sich aus dem Motion Update, welches die Roboterposition bestimmt und dem Measurement Update zur Verbesserung der Position zusammen. Hierbei wird das Motion Update ebenso wie in Feature basierten SLAM-Verfahren berechnet. Das Measurement Update unterscheidet sich hingegen zwischen dem Grid Map basierten Ansatz, der größtenteils betrachtet wird, und dem Feature basierten Ansatz.

4.5.1 Motion Update

Die Berechnung basiert auf dem Extended Kalman Filter und bestimmt aus einer gegebenen Mittelwert Roboterposition μ_{t-1} mit der dazugehörigen Kovarianzmatrix Σ_{t-1} und dem Bewegungsvektor u_t die neue Roboterposition. Dabei wird auch die Kovarianzmatrix angepasst. Das (I) Velocity- und das (II) Odometrie Modell unterscheiden sich in ihrer Berechnung.

I. Velocity Modell wird nach der Berechnungsvorschrift von Thrun et al. [TFB06], welche im EKF-SLAM beschrieben ist, ermittelt. G_t ist eine Jacobian Matrix, die durch das Ableiten nach μ_{t-1} des Bewegungsvektors

$$\mu_t = \mu_{t-1} + \begin{pmatrix} -\frac{\nu}{\omega} \sin \theta + \frac{\nu}{\omega} \sin (\theta + \omega \Delta t) \\ \frac{\nu}{\omega} \cos \theta - \frac{\nu}{\omega} \cos (\theta + \omega \Delta t) \\ \omega \Delta t \end{pmatrix} = g(u_t, \mu_{t-1})$$

ermittelt wird.

$$G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}} = \begin{pmatrix} 1 & 0 & -\frac{\nu}{\omega} \cos \mu_\theta + \frac{\nu}{\omega} \cos (\mu_\theta + \omega \Delta t) \\ 0 & 1 & -\frac{\nu}{\omega} \sin \mu_\theta + \frac{\nu}{\omega} \sin (\mu_\theta + \omega \Delta t) \\ 0 & 0 & 1 \end{pmatrix}$$

Die Bewegungsfehler werden in der Kovarianzmatrix M_t ermittelt.

$$M_t = \begin{pmatrix} \alpha_1 \nu_t^2 + \alpha_2 \omega_t^2 & 0 \\ 0 & \alpha_3 \nu_t^2 + \alpha_4 \omega_t^2 \end{pmatrix}$$

$$V_t = \begin{pmatrix} \frac{-\sin \theta + \sin(\theta + \omega_t \Delta t)}{\omega_t} & \frac{\nu_t(\sin \theta - \sin(\theta + \omega_t \Delta t))}{\omega_t^2} + \frac{\nu_t(\cos(\theta + \omega_t \Delta t) \Delta t)}{\omega_t} \\ \frac{\cos \theta - \cos(\theta + \omega_t \Delta t)}{\omega_t} & -\frac{\nu_t(\cos \theta - \cos(\theta + \omega_t \Delta t))}{\omega_t^2} + \frac{\nu_t(\sin(\theta + \omega_t \Delta t) \Delta t)}{\omega_t} \\ 0 & \Delta t \end{pmatrix}$$

Die Kovarianzmatrix Σ_{t-1} wird mit folgender Gleichung aktualisiert:

$$\Sigma_t = G_t \Sigma_{t-1} G_t^T + V_t M_t V_t^T$$

II. Odometrie Modell Berechnung ist dem Velocity Modell nachempfunden. Dabei wird die Herleitung des Velocity Modells auf das Odometrie Modell angewendet und folgende Berechnung ermittelt. Als Grundlage dient die Berechnung der Roboterposition μ_t :

$$\mu_t = \mu_{t-1} + \begin{pmatrix} \delta_{trans} \cos (\theta + \delta_{rot1}) \\ \delta_{trans} \sin (\theta + \delta_{rot1}) \\ \delta_{rot1} + \delta_{rot2} \end{pmatrix} = g(u_t, \mu_{t-1})$$

Daraus wird die Jacobian Matrix G_t abgeleitet:

$$G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}$$

$$G_t = \begin{pmatrix} 1 & 0 & -\delta_{trans} \sin(\theta + \delta_{rot1}) \\ 0 & 1 & \delta_{trans} \cos(\theta + \delta_{rot1}) \\ 0 & 0 & 1 \end{pmatrix}$$

Die Bewegungsfehler werden in diesem Fall als 3x3 Matrix M_t dargestellt, da drei Bewegungsparameter verwendet wurden:

$$M_t = \begin{pmatrix} \alpha_1 \delta_{rot1}^2 + \alpha_2 \delta_{trans}^2 & 0 & 0 \\ 0 & \alpha_3 \delta_{trans}^2 + \alpha_4 \delta_{rot1}^2 + \alpha_4 \delta_{rot2}^2 & 0 \\ 0 & 0 & \alpha_1 \delta_{rot2}^2 + \alpha_2 \delta_{trans}^2 \end{pmatrix}$$

Die Jacobian Matrix V_t lässt sich folgendermaßen berechnen:

$$V_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial u_t}$$

$$V_t = \begin{pmatrix} -\delta_{trans} \sin(\theta + \delta_{rot1}) & \cos(\theta + \delta_{rot1}) & 0 \\ \delta_{trans} \cos(\theta + \delta_{rot1}) & \sin(\theta + \delta_{rot1}) & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

Die Kovarianzmatrix Σ_{t-1} wird mit derselben Gleichung wie im Velocity Modell aktualisiert:

$$\Sigma_t = G_t \Sigma_{t-1} G_t^T + V_t M_t V_t^T$$

4.5.2 Measurement Update

Das Ziel des Measurement Update ist es, die Roboterposition mithilfe der Sensordaten zu präzisieren und die Gaußsche Glockenkurve, welche durch die Kovarianzmatrix beschrieben wird und die Ungenauigkeit angibt, zu strecken. Hierfür existieren in wissenschaftlichen Arbeiten mehrere Varianten, jedoch gibt es nur wenige Umsetzungen in Verfahren.

Improved Proposal Distribution ist ein Verfahren aus Grisetti et. al. [GSB], das versucht eine bessere Verteilung vorzuschlagen, indem es im Umkreis der Roboterposition einige weitere Positionen bestimmt und von diesen dann die Wahrscheinlichkeit berechnet. Über den Positionen und deren Wahrscheinlichkeiten wird nun interpoliert und eine finale Roboterposition und die dazugehörige Kovarianzmatrix ermittelt.

Der Hector-SLAM besitzt eine solche Implementierung und setzt diese um, indem eine Abwei-

chung

$$\Delta = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

definiert wird. Damit werden 6 neue Punkte erzeugt, die mit dem aktuellen Punkt in den Spalten der Matrix

$$\delta_{\text{Punkte}} = \begin{pmatrix} x + \Delta_x & x - \Delta_x & x & x & x & x & x \\ y & y & y + \Delta_y & y - \Delta_y & y & y & y \\ \theta & \theta & \theta & \theta & \theta + \Delta_\theta & \theta - \Delta_\theta & \theta \end{pmatrix}$$

eingetragen werden.

Zu jedem dieser Punkte wird ein Vektor p mit den Wahrscheinlichkeiten p_1 bis p_7 der einzelnen Punkte, die durch ein Wahrscheinlichkeitsmodell erzeugt wurden, erstellt. Die neue Roboterposition μ berechnet sich nun aus

$$\begin{aligned} \mu = & \begin{pmatrix} x + \Delta_x \\ y \\ \theta \end{pmatrix} * p_1 + \begin{pmatrix} x - \Delta_x \\ y \\ \theta \end{pmatrix} * p_2 + \begin{pmatrix} x \\ y + \Delta_y \\ \theta \end{pmatrix} * p_3 + \begin{pmatrix} x \\ y - \Delta_y \\ \theta \end{pmatrix} * p_4 + \\ & \begin{pmatrix} x \\ y \\ \theta + \Delta_\theta \end{pmatrix} * p_5 + \begin{pmatrix} x \\ y \\ \theta - \Delta_\theta \end{pmatrix} * p_6 + \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} * p_7 \\ & \eta = \frac{1}{\sum_{i=1}^7 p_i} \\ & \mu = \mu * \eta \end{aligned}$$

Die dazugehörige Kovarianzmatrix berechnet sich durch

$$\Sigma = \sum_{i=1}^7 p_i * \eta * (\delta_i - \mu) * (\delta_i - \mu)^T$$

Leider gibt es in der Implementierung des Hector-SLAM keine Stelle, in der die beschriebene Funktionalität verwendet wird.

Gaussian sum Repräsentation (SoG) wird in der Arbeit von Nieto et. al. [NBN] beschrieben. Das Verfahren nimmt zwei horizontale Laserscans in denen jeder Punkte eine Position und eine Varianz besitzt, aus denen dann jeweils eine Gaußsche Summe gebildet wird. Nachdem das Kreuzprodukt aus den beiden gebildet wurde, können die Roboterposition sowie die Kovarianzmatrix berechnet werden.

Es wurde leider keine Umsetzung in den Implementierungen entdeckt, aber in der Arbeit [NBN] ist die Vorgehensweise gut beschrieben.

4.6 POSITIONSKORREKTUR

Das Ziel der Positionskorrektur besteht darin, die Position des Roboters auf der Karte mithilfe der Sensordaten so exakt wie möglich zu bestimmen. Hierbei sind die Odometriedaten sowie die Fehlerinformationen des Roboters irrelevant, da die gefundenen Algorithmen allein anhand der Sensordaten die Position bestimmen. Mithilfe der Odometriedaten können nur Hilfestellungen geleistet werden, indem Algorithmen wie der ICP ein kleineres Suchfeld bzw. einen genaueren Suchbereich für die Übereinstimmung vorgegeben bekommt.

Ansonsten ist das Ziel der Korrekturverfahren die Sensordaten so exakt wie möglich auf die Kartendaten abzubilden oder eine möglichst genaue Transformation zwischen jeweils zwei Sensor Datensätzen zu ermitteln. Folgende Algorithmen werden in den genannten SLAM-Verfahren verwendet:

ICP verwendet Tiefeninformationen aus Laserscannern oder Tiefenbildkameras. Hierbei ermittelt der Algorithmus zwischen den Sensordaten und Kartendaten eine Transformation mit der die Sensordaten in das Koordinatensystem der Karte transformiert werden. In diesem Verfahren bestehen die Daten aus einer Punktmenge aus R^n , es kann aber auch eine Menge von Linien, Dreiecken, Oberflächen, etc. sein. Das Modell \mathcal{M} ist in unserem Fall die Karte und die Szene \mathcal{P} sind die Sensordaten. Der ICP Algorithmus benötigt folgende Methoden für die Berechnung der Transformation von \mathcal{P} in \mathcal{M} :

1. Die Registrierungsmethode berechnet zu einem Punkt $\vec{p} \in \mathcal{P}$ einen passenden Punkt in \mathcal{M} . Das kann für alle Punkte geschehen, es muss jedoch für eine Mindestmenge durchgeführt werden. Im 2-dimensionalen Raum werden mindestens 2 Punkte und im 3-dimensionalen Raum mindestens 3 Punkte mit der Bedingung, dass sie nicht auf einer Geraden liegen, benötigt.

Die Registrierung von \mathcal{P} in \mathcal{M} kann durch die Verwendung einer Distanzfunktion durchgeführt werden, indem der nächste Nachbar aus \mathcal{M} zum Punkt \vec{p} gesucht wird. In diesem Fall besteht die Möglichkeit einer komplett falschen Zuordnung, dies wird jedoch durch die Verwendung zusätzlicher Merkmale wie Umgebungsinformationen der Punkte erweitert und verbessert. Ein Beispiel wäre der Levenberg-Marquardt Algorithmus, welcher in das MRPT-Framework integriert ist.

2. Die Minimierungsmethode berechnet eine Rotation und Translation, welche die in der Registrierungsmethode gefundenen Punktpaare iterativ aufeinander abbildet. In dem SLAM 6D aus der OpenSLAM Plattform sind 10 verschiedene Minimalisierungsalgorithmen implementiert. Ein oft verwendeter ist die Singulärwertzerlegung SVD (Singular Value Decomposition). Weitere Algorithmen wären z.B. Linearisierung mit eulerschen Winkeln, Linearisierung mit Quaternionen, Quaternionen, Dual Quaternionen, Quaternionen und Skalierungsfaktor, Helix Transformation, kleine Winkel Approximation, orthonormale Matrizen und Approximation mithilfe von Normalen der Punkte.

3. Die Transformation wendet die in der Minimierungsmethode ermittelte Transformation auf \mathcal{P} an.

Der ICP Ablauf sieht vor, dass die Schritte 1 bis 3 so oft wiederholt werden bis entweder eine maximale Iterationsanzahl erreicht ist oder die Distanz von \mathcal{P} und \mathcal{M} bzw. die Distanzverbesserung

zwischen zwei Iterationen hinreichend klein ist.

SIFT ist ein Feature Detektor Verfahren, das auf Bilder angewendet wird. Es wurde speziell für Graustufenbilder entwickelt. Dieser Algorithmus ermittelt in einem Bild Features und zu den Features einen Beschreibungsvektor, der Informationen über die Umgebung und den Skalierungsfaktor des Features enthält. Mit diesen Beschreibungsvektoren besteht die Möglichkeit Bilder untereinander zu vergleichen und Features zu identifizieren, die in mehreren Bildern vorkommen.

Mithilfe dieser Informationen können Transformationen zwischen den Bildern und somit die Roboterbewegung zwischen den Sensoraufnahmen ermittelt werden. Eine weitere Möglichkeit zur Verwendung von SIFT zeigt sich im Zusammenhang mit RGBD-Aufnahmen und dem ICP Algorithmus, indem mittels SIFT die ICP Zuordnungen bzw. Schätzungen dieser ermittelt werden und mit diesen Informationen der ICP schneller und genau arbeiten kann.

4.7 RESAMPLING

Das Resampling ist eine Komponente, die nur bei einem partikelbasierten Ansatz benötigt wird. Das Ziel des Resampling besteht darin, dass eine Liste von Wahrscheinlichkeitswerten überarbeitet wird, dabei die unwahrscheinlichen Werte entfernt und sehr wahrscheinliche Werte vervielfacht werden.

Es werden ein Array an Wahrscheinlichkeitswerten und die Partikelanzahl nach dem Resampling übergeben, wobei meist die Ausgabe Partikelanzahl die gleiche ist, wie die Anzahl der übergebenen Partikel Wahrscheinlichkeiten. Das Ergebnis dieses Verfahrens ist ein Array mit der definierten neuen Partikelanzahl als Länge und jede Array Position besitzt nun den Wert zu einer Position im Eingabearray. Dabei können Werte mehrfach vorkommen.

In den Implementierungen der SLAM-Verfahren gibt es 4 verschiedene Resampling Algorithmen. Dies wären das Multinomial, Residual, Systematic und Stratified Resampling. Alle Algorithmen haben die gleichen Eingabe- und Ausgabeparameter.

Multinomial Resampling ist die einfachste Resampling Methode und basiert auf der Multinomialverteilung, welche auf dem Modell *ungeordnetes Ziehen aus einer Urne mit Zurücklegen* basiert.

In der Dokumentation zum MRPT-Framework [mrp] wird es mithilfe der Rad Analogie beschrieben, welche im Folgenden eingesetzt wird um den Algorithmus und das Vorgehen zu beschreiben. Zu Beginn müssen die Wahrscheinlichkeiten ω_i mit $i = 1 \dots N$ der Partikel normalisiert werden, worauf sie wie in Abbildung 4.6 zu sehen ist hintereinander auf dem Rad angeordnet werden.

Dann werden M Zufallszahlen im Bereich $[0, 1)$ gebildet, wobei M die Anzahl der gesuchten Partikel ist. Jede Zufallszahl repräsentiert nun eine Partikelposition auf dem Kreis, der nur noch ein Partikel ω_i zugeordnet werden muss.

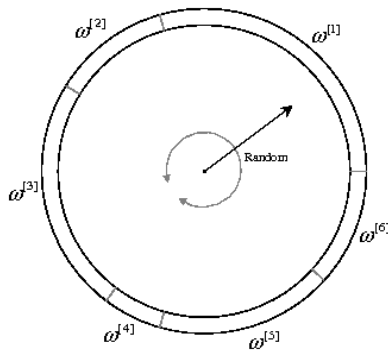


Abbildung 4.6: Multinomial Resampling Rad
Analogie aus der Dokumentation
des MRPT-Frameworks

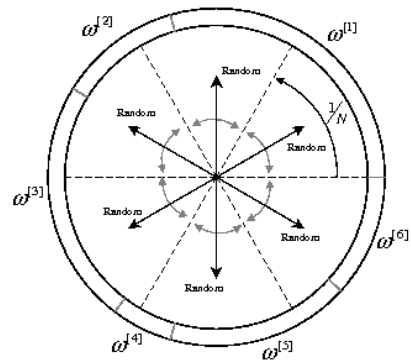


Abbildung 4.7: Stratified Resampling Rad
Analogie aus der Dokumentation
des MRPT-Frameworks

In der Implementierung kann dieses Verfahren, wie z.B. im MRPT-Framework im Pseudocode Listing 4.4 umgesetzt werden. Die Besonderheit in dem Algorithmus ist das Verwenden der kumulativen Summe um den Gewichte im Bereich von 0.0 bis 1.0 Positionen zuzuordnen, um in der WHILE-Scheife eine Zuordnung zwischen den Zufallswerten und den Gewichten zu erhalten.

Listing 4.4: Multinomial Resampling im MRPT-Framework

```

N = Eingabe Partikelanzahl;
M = Ausgabe Partikelanzahl;
double[N] Weights = Array mit normalisierten Partikel Gewichten;
double[N] Q = Cumsum ( kumulative Summe) von Weights;
double[M] Randoms = Randomwerte zwischen 0.0 und 1.0;
Randoms.sort();
i = j = 0;
double[M] ergebnis;
while ( i < M ) {
    if ( Randoms[ i ] < Q[ j ] ) {
        ergebnis[ i++ ] = j;
    }
    else {
        j++;
        if ( j >= M ) {
            j = M - 1;
        }
    }
}
return ergebnis;

```

Residual Resampling ist im Vergleich zum Multinomial Resampling nicht mehr komplett zufallsbasiert. Es fügt zu Beginn die höchst gewichteten Partikel zu der Zielmenge hinzu und füllt die letzten fehlenden Partikel durch Multinomial Resampling auf.

Das Vorgehen ist dementsprechend, dass in der Vorverarbeitung alle Partikel mit einem Gewicht von $\omega_i \geq \frac{1}{M}$ (M ... Partikelanzahl der Eingabe) genau $\lfloor M\omega_i \rfloor$ ($i \in 1 \dots M$) mal in die Zielpartikel-liste aufgenommen werden.

Wenn die Zielpartikelliste ihre Gesamtanzahl an Partikel noch nicht erreicht hat, wird eine Neugewichtung der Partikel

$$\omega[i]_{neu} = \frac{1}{\text{fehlende Partikelanzahl}} (M\omega[i] - \text{Anzahl Partikel } i \text{ in Zielpartikelliste})$$

durchgeführt.

Für die fehlende Anzahl an Partikeln in der Zielliste wird nun auf die neue Liste der Gewichte $\omega[i]_{neu}$ ($i \in 1 \dots M$) das Multinomial Resampling wie zuvor erklärt durchgeführt.

Stratified Resampling ist eine Abwandlung des Multinomial Resampling. Wie in der Abbildung 4.7 zu sehen ist, wird das Rad in M (Anzahl Eingabepartikel, wird in der Abbildung als N bezeichnet) gleichgroße Teile unterteilt. In jedem der Teile sucht ein Zufallsgenerator eine Zahl für das Multinomial Resampling, wobei damit der Unterschied zum normalen Multinomial Resampling darin besteht, dass die Zufallswerte perfekt über das komplette Rad verteilt sind und keine Bereiche vernachlässigt werden.

Nach der Zufallswertbestimmung wird wie im Algorithmus 4.4 in der WHILE-Schleife das Resampling durchgeführt.

Systematic Resampling wird auch universelles Resampling genannt und basiert ebenso auf dem Multinomial Resampling. Die Zufallszahlen werden jedoch anders bestimmt. Bei diesem Verfahren wird eine Zufallszahl im Bereich $[0; \frac{1}{M})$ per Zufallsgenerator bestimmt. Alle weiteren Werte des Random Arrays im Algorithmus 4.4 werden erzeugt durch:
 $\text{Randoms}[i] = \text{Zufallszahl} + i \frac{1}{M}$.

Nachdem die Zufallswerte im Random Array alle bestimmt sind, wird wie im Algorithmus 4.4 in der WHILE-Schleife das Resampling durchgeführt.

4.8 KARTEN

Es gibt verschiedene Arten von Kartendarstellungen für SLAM-Verfahren, diese sind u.a. Grid Map, Octo Map, Point Cloud Map, Feature Map und Graph Map. Jedoch werden für die in dieser Arbeit vorgestellten SLAM-Verfahren mit Laserscannern und Tiefenbildkameras nur Grid Maps, Octo Maps und Point Cloud Maps benötigt.

Der Großteil der Verfahren, wie z.B. Hector-SLAM, Tiny SLAM, Grid SLAM, ICP SLAM und RBPF SLAM verwendet die Grid Map, da sie hauptsächlich auf horizontalen Laserscans basieren. Die Verfahren SLAM 6D und RGBD-SLAM verwenden eine Octo Map. Die Point Cloud Map wäre ebenfalls eine Möglichkeit, findet aber in den angegebenen Verfahren keine Anwendung.



Abbildung 4.8: Darstellung eines Raumes in einer Occupancy Grid Map [TFB06]



Abbildung 4.9: Darstellung eines Raumes in einer Occupancy Grid Map [TFB06]

Tabelle 4.1: Vergleich der Kartentypen

Kartentyp	Speicherbedarf	Dimension	Kartenanzahl	Speicherzunahme
Grid MAP	fest definiert	2D	eine	linear mit Umgebung
Octo Map	fest definiert	3D	eine	linear mit Umgebung
Point Cloud Map	unbestimmt	3D	eine	linear mit Sensoraufnahme
DP Map	Maximum ist fest definiert	2D	mehrere	linear mit Umgebung

Es gibt noch einen Spezialfall für eine Grid Map, das ist die DP-Map aus dem DP-SLAM. Diese Karte wurde speziell für Partikelfilter entwickelt um eine Speicherplatz Einsparung zu erreichen. Die Tabelle 4.1 enthält die Eigenschaften der einzelnen Kartentypen.

Grid Map ist eine Karte, die auf Zellen basiert, welche einen der drei Zustände *frei - free* (Bereich kann betreten werden), *belegt - occupied* (Bereich kann nicht betreten werden, da dort ein Hindernis wie z.B. eine Wand ist) oder *unbekannt - unknown* (Bereich wurde noch nicht von den Sensoren aufgenommen) hat. In den Implementierungen wird unbekannte Zellen der Wert 0.5, freie Zellen ein Wert > 0.5 (meist 1.0) und belegten Zellen ein Wert < 0.5 (meist 0.0) zugewiesen. In der Abbildung 4.8 ist eine Grid Map mit einem Flur zu sehen, durch die kleine Fläche sind die Zellen gut zu erkennen. Die Abbildung 4.9 zeigt eine Karte des San Jose Tech Museums mit einem gut erkennbaren Grundriss.

In den Implementierungen bekommen die Werte meist als Datentyp *float* mit einem Speicherbedarf von nur 4Byte im Vergleich zu *double* mit 8Byte zugewiesen. Die Größe jeder Zelle ist durch ihre Kantenlänge vordefiniert. Die Zellen sind quadratisch mit einer Kantenlänge von 15cm oder weniger, je nachdem wie die Umgebung und Genauigkeit gewünscht wird. Außerdem sind die Werte in einem eindimensionalen Array verwaltet, das zu Beginn auf die Anfangsgröße der Um-

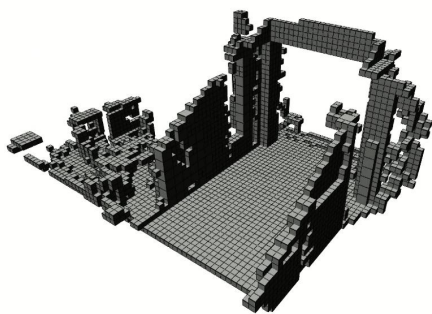


Abbildung 4.10: Darstellung eines Raumes in einer Octo Map [Hor13]



Abbildung 4.11: Darstellung eines Raumes in einer Octo Map mit eingefärbten Zellen [Hor13]

gebung mit einer definierten Breite und Länge initialisiert wird. Mit zunehmender Umgebungsgröße ist eine Reorganisation des Arrays nötig.

Octo Map ist ebenso wie die Grid Map eine Zellen basierte Karte, jedoch mit dem Unterschied, dass die Zellen 3-dimensional und würfelförmig sind. Die Definition der Zellen ist genauso wie die der Grid Map. Ein Beispiel ist in den Abbildungen 4.10 und 4.11 zu sehen. In der Implementierung erhält das Array eine 2. Dimension für die Höhe.

Point Cloud Map ist eine Punktwolke, die eine Liste von allen aufgenommenen Punkten des Tiefensensors verwaltet. Mit jeder Aufnahme steigt der benötigte Speicherplatz an, da die Punkte oftmals nicht aufeinander liegen. Wenn die Aufnahmen sich stark überlappen, nimmt dieser Effekt weiter zu, da auf einem kleinen Bereich immer mehr Punkte existieren. Um diesem Effekt entgegenzuwirken, sind zusätzliche Algorithmen zur Speicherreduzierung nötig.

Die Point Cloud Map ist eine Alternative zur Octo Map, jedoch wird sie aufgrund ihrer Nachteile in keiner Implementierung verwendet. Es gibt eine Abwandlung einer Point Cloud Map, in welcher ein Graph aufgebaut wird, der zu jeder Roboterposition eine Punktwolke des Sensors und damit keine gesamte Karte sondern nur Teilaufnahmen der Umgebung mit ihren Transformationen untereinander abspeichert.

Distributed Partikel Map ist wie oben beschrieben eine Grid Map, jedoch ist die Karte für den Partikelfilter Einsatz konzipiert.

In einer DP-Map wird nicht nur eine Karte, sondern es werden alle Karten einer Partikelliste verwaltet und gespeichert. Dabei baut das Verfahren eine Baumstruktur für die verschiedenen Zellen der Grid Map auf, in welchen ihre Partikelzugehörigkeit gespeichert wird. Damit werden Zellen, die in mehreren Partikeln vorkommen, nur einmal gespeichert und somit eine Speicherplatzersparnis durch höheren Rechenaufwand erreicht. Außerdem müssen keine kompletten Kopien von Karten mehr erstellt werden, wie es bei der Partikelduplizierung sonst der Fall ist.

Partikel Map Liste (Partikelfilter Map) ist eine Komponente, die im Zusammenhang mit dieser Arbeit eingeführt wurde um die Distributed Partikel Map optimal in das Konzept zu integrieren.

Wenn ein Partikelfilter basiertes SLAM-Verfahren verwendet werden soll, kann entweder die Distributed Partikel Map oder eine der anderen Kartentypen verwendet werden. Da aber die anderen Kartentypen nur jeweils eine Karte repräsentieren, können sie nicht als Alternative für eine Distributed Partikel Map verwendet werden und umgedreht genauso. An dieser Stelle setzt die Partikelfilter Map ein, indem sie als Schnittstelle zu den einzelnen Partikeln dient und alle Partikel Karten speichert und die benötigten Funktionen und Operationen für den Zugriff auf einzelne Partikel bereitstellt.

Somit kann die Distributed Partikel Map einfach integriert und die Verwendung anderer Karten ersetzt werden.

4.9 ZUSAMMENFASSUNG

In diesem Abschnitt wurde ein Überblick über die verschiedenen Bestandteile Grid basierter SLAM-Verfahren gegeben. Die gebildeten Komponenten sind in der Abarbeitung je nach verwendetem SLAM Algorithmus austauschbar bzw. irrelevant. Ein Hauptkriterium ist dabei, dass die verschiedenen Komponenten eigenständig verwendet werden können und die Möglichkeit besteht sie durch andere zu ersetzen. Hierbei spielen die Datenformate sowie die Existenz alternativer Algorithmen eine wichtige Rolle. In diesem Zusammenhang wurden für die Komponenten mehrere mögliche Algorithmen aufgezeigt bzw. erklärt.

Aus diesen extrahierten Komponenten wird im folgenden Abschnitt 5 ein Feature Modell erstellt, anhand dessen der Anwender verschiedene SLAM-Verfahren zusammenstellen kann. Beispiele für die Zusammenstellung einiger SLAMs aus Abschnitt 3.1 werden ebenso anhand des Feature Diagramms aufgezeigt.

Die extrahierten Komponenten und deren mögliche Funktionen sind jedoch nur ein Überblick der SLAM-Verfahren, die für diese Arbeit analysiert wurden. Es gibt noch weitere Implementierungsmöglichkeiten.

5 FEATURE MODELL FÜR SLAM-VERFAHREN

In den vorherigen Abschnitten dieser Arbeit wurde ein Überblick über die wichtigsten Grundbegriffe zum Thema SLAM gegeben, einige SLAM-Verfahren mit ihren Funktionsweisen sowie Voraussetzungen erklärt und daraus eine Liste an Komponenten aufgebaut. Diese beschreiben die verschiedenen Bestandteile der SLAM Algorithmen und stellen verschiedene Funktionalitäten bereit. Anhand dieser Komponenten besteht die Möglichkeit ein SLAM-Verfahren zusammenzusetzen.

Im Folgenden wird eine Einführung in den Aufbau und die Darstellungsweise eines Feature Diagramms gegeben. In dieser Darstellungsform werden daraufhin die extrahierten Komponenten zusammentragen und ein Featurebaum aufgespannt. Im Abschnitt 5.3 werden Feature Diagramm Repräsentationen anhand einiger zuvor erklärter SLAM-Verfahren dargestellt.

5.1 FEATURE DIAGRAMM

Feature Modelle werden seit den frühen 1990er Jahren verwendet um eine Anwendungsdomain mit ihren Features, deren Verwendbarkeit und Variabilität zu beschreiben. Seitdem verwendet man sie größtenteils für die Entwicklung von Software Produktlinien, um damit modellgetriebene Software zu generieren.

Ein Feature Diagramm ist ein hierarchisch aufgebauter Baum mit einem Wurzelknoten, wobei die Knoten durch Features repräsentiert werden.

Feature *ist ein unterscheidbares Merkmal des Softwareprodukts, das z.B. ein System, eine Komponente, ein Algorithmus sein kann.*

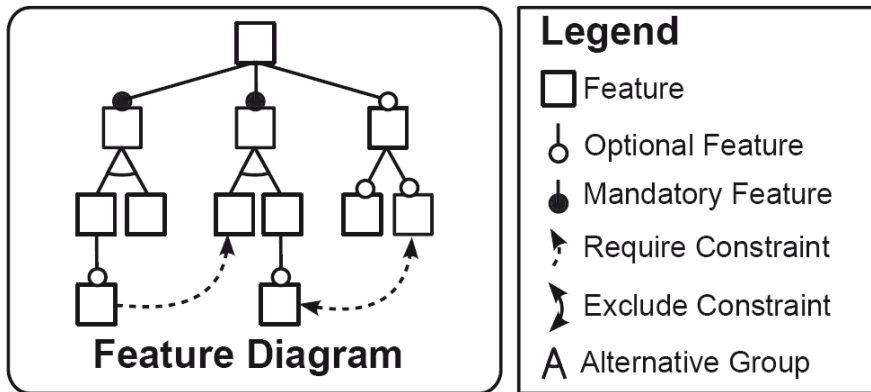


Abbildung 5.1: Beispiel Feature Diagramm mit der dazugehörigen Legende aller möglichen Elemente [Sch13]

In Abbildung 5.1 ist auf der linken Seite ein Beispiel Feature Diagramm mit der dazugehörigen Legende auf der rechten Seite dargestellt. Für die Verbindung der Features untereinander gibt es mehrere Varianten, die erste ist die Angabe, ob ein Feature optional (Optional Feature) oder verpflichtend (Mandatory Feature) verwendet werden muss. Features können weiterhin präzisiert werden, indem alternative Gruppen (Alternative Group) bzw. beliebige Gruppen (Or Group) Verwendung finden. Weiterhin gibt es noch Bedingungen, die Relationen zwischen den Features beschreiben. Das können ausschließende Beschränkungen (Exclude Constraint) oder benötigte Beschränkungen (Require Constraint) sein.

5.2 SLAM FEATURE DIAGRAMM

Im Folgenden wird ein Feature Modell zum Erstellen verschiedener SLAM-Verfahren entwickelt, das auf den bisher behandelten Bereich von SLAM-Verfahren, welche Tiefenbild bzw. Laserscanner Daten verwenden, ausgelegt ist. Das Ziel ist eine Vielzahl verschiedener SLAM-Verfahren aus diesem Anwendungsbereich zu erzeugen und somit eine einfache Anpassbarkeit der Algorithmen an verschiedene Anwendungsfälle bzw. Systemvoraussetzungen zu erreichen. Außerdem erhält man eine Übersicht über die existierenden Teile der SLAM-Verfahren und den daraus resultierenden Abhängigkeiten untereinander, wodurch der Entwickler ein einfaches Werkzeug für die Erstellung von SLAM-Verfahren bekommt.

Das Diagramm mit dem Feature Baum ist in Abbildung 5.2 dargestellt. Der Wurzelknoten ist der *SLAM*, welcher sich aus den verpflichtenden Features (Map, Sensor Data und Sensors) zusammensetzt, ohne die kein SLAM-Verfahren arbeiten kann. Die optionalen Features (Motion Data, EKF, Particle filter, Feature generation, Position estimate und Likelihood Model) können je nach SLAM-Verfahren verwendet oder ignoriert werden.

In den tieferen Baumebenen gibt es *alternative* und *or* Verknüpfungen, die dem Anwender vorschreiben, ob er sich für eines entscheiden muss (*alternative*) oder die Wahl zwischen allen hat (*or*). Das ist z.B. im *EKF* unter *Motion update*, da in diesem Fall nur ein Bewegungsmodell verwendet werden kann. Die Wahl des Modells hängt dabei auch von den *Motion Data* ab, da jedes Bewegungsmodell eine bestimmte Bewegungsdatenrepräsentation benötigt (*require*). Neben den

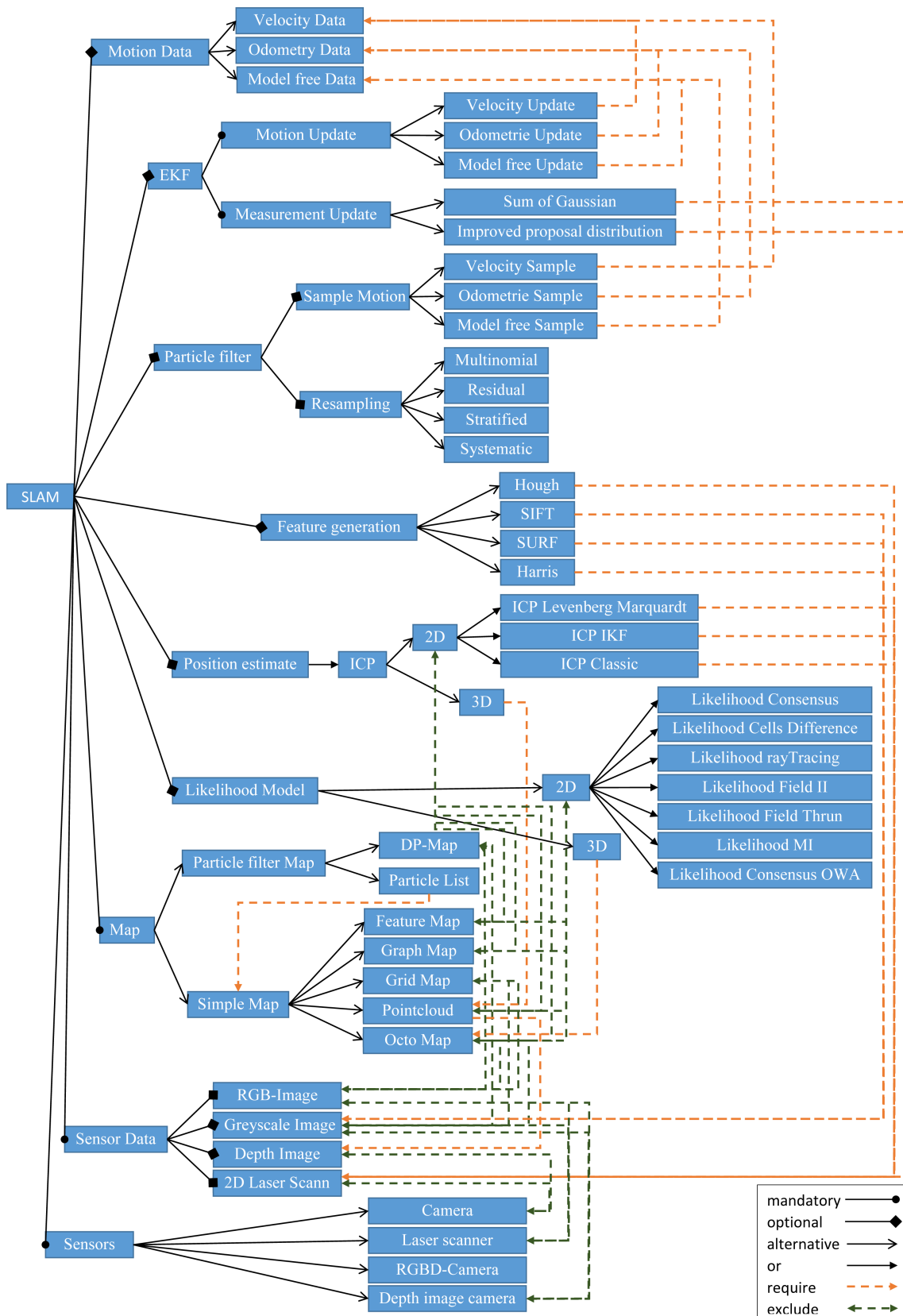


Abbildung 5.2: Feature Diagramm Darstellung für SLAM-Verfahren

benötigten (require) Features gibt es noch ausschließende (exclude) Features, welche bei den *Sensors*, *Sensor Data* und *Map* zu erkennen sind. Es wird z.B. ausgesagt, dass die Verwendung einer Kamera die Existenz der Sensordaten in Form eines Tiefenbildes bzw. 2-dimensionalen Laserscans ausschließt. Somit kann kein RGB-Bild als Sensordatum existieren, welches wiederum die Existenz einer Grid Map und Octo Map aufgrund der ausschließenden Relation bzw. die Point Cloud Map, da diese ein Tiefenbild benötigt (require).

Weitere Besonderheiten des Feature Baums werden im nächsten Abschnitt im Zusammenhang mit den Beispieldarstellungen der SLAM-Verfahren erläutern.

5.3 UMSETZUNG AUSGEWÄHLTER SLAM-VERFAHREN

Im Folgenden werden SLAM-Verfahren aus dem Abschnitt 3.1 als Feature Diagramm dargestellt. Die Auswahl der SLAMs wurde dementsprechend getroffen, dass anhand dieser Besonderheiten des Diagrammaufbaus, wie die optionalen Features im Partikelfilter aufgrund des Core-SLAM, erklärt werden können. Weiterhin sollen die Unterschiede zwischen einem SLAM mit und ohne Odometriedaten hervorgehoben werden, sowie die annähernd identischen SLAM-Verfahren trotz unterschiedlichen Bezeichnungen am Beispiel des RBPF und GMapping SLAM. In den Feature Diagrammen zu den speziellen SLAM Umsetzungen sind die verwendeten Features grün markiert.

Core-SLAM (siehe Abbildung 5.3) verwendet einen *Laser scanner*, wodurch ein *2D Laser Scann* als Sensordaten dem SLAM-Verfahren bereitgestellt wird. Weiterhin werden dem SLAM Bewegungsdaten in Form von *Velocity Data* zur Berechnung der Roboterposition bereitgestellt. Das Verfahren ist kein partikelbasiertes, jedoch werden mehrere mögliche Roboterpositionen bestimmt und von diesen die wahrscheinlichste weiterverwendet, wofür die *Velocity Sample* Berechnung benötigt wird. Da aber kein Resampling erforderlich ist, werden die Folgeknoten von *Particle filter* nur als optionale Features angeboten. Im Weiteren verwendet das SLAM-Verfahren eine *Grid Map* sowie das *Likelihood Cells Difference* Feature.

GMapping SLAM (siehe Abbildung 5.4) ist ein Rao-Blackwellized Partikelfilter basiertes Verfahren, das den Großteil der bereitgestellten Features des *SLAM* Wurzelknoten verwendet. Es nutzt ebenso wie der Core-SLAM die Bewegungsinformationen des Roboters sowie einen *2D Laser Scann* als Sensordaten. Diese Daten werden jedoch aus einer *RGBD-Camera* generiert. Da es ein partikelbasiertes Verfahren ist, werden alle Features des *Particle filter* Knoten sowie eine *Particle filter Map* benötigt. Hinzu kommt der *EKF* mit seinen Features, da hiermit eine Präzisierung der einzelnen Partikel durchgeführt wird.

RBPF-SLAM (siehe Abbildung 5.5) unterscheidet sich nur geringfügig vom GMapping SLAM, da sie in ihrer Funktionsweise gleich sind. Der RBPF-SLAM ist jedoch im MRPT-Framework integriert, wodurch mehrere Algorithmen für einzelne Features bereitgestellt werden. Deshalb sind in den Diagrammen nur Unterschiede im verwendeten *Resampling* sowie *Likelihood Model* Feature zu erkennen. Es besteht hier ebenso die Möglichkeit eine *RGBD-Camera* zu verwenden.

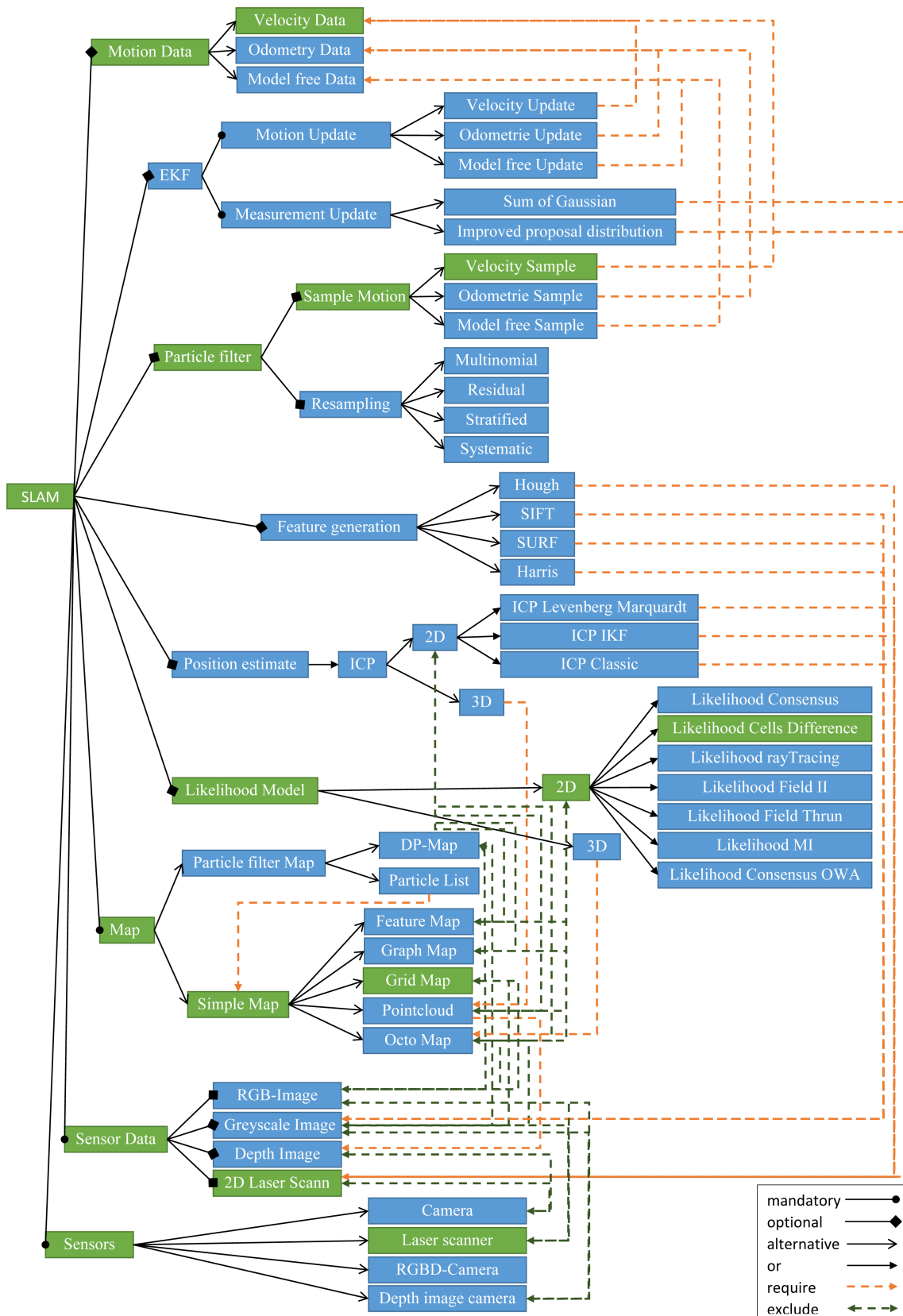


Abbildung 5.3: Feature Diagramm des Core-SLAM

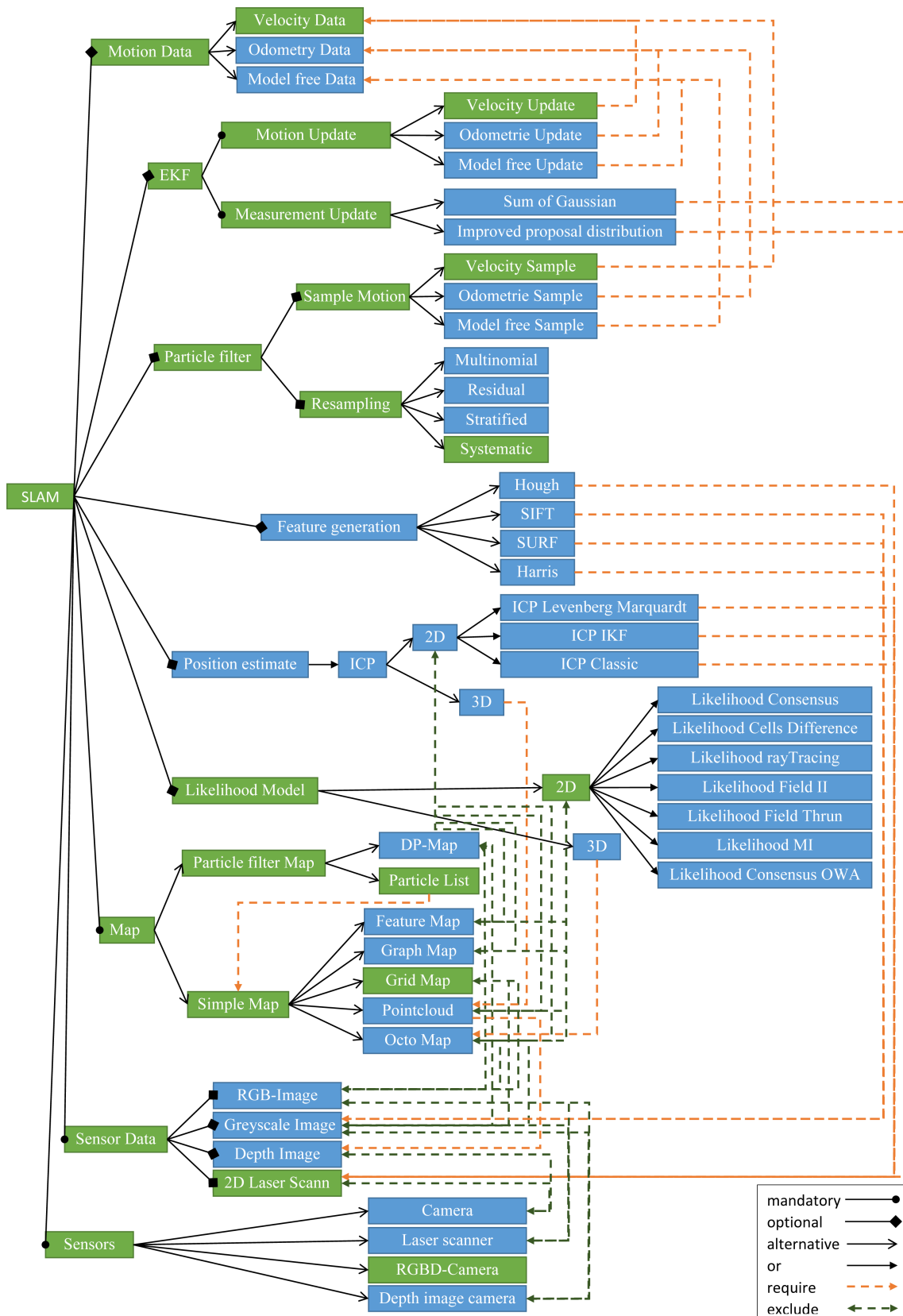


Abbildung 5.4: Feature Diagramm des GMapping SLAM

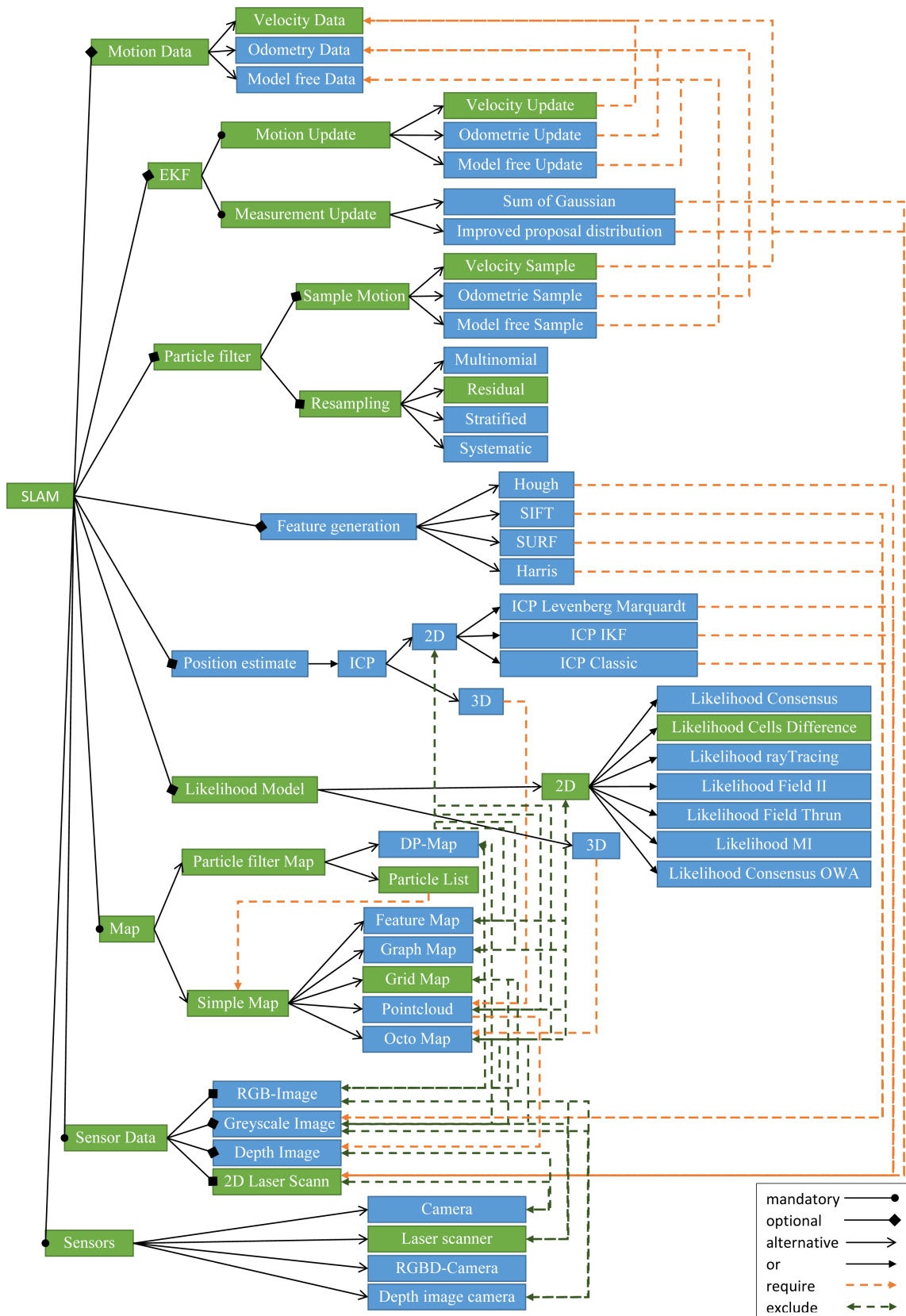


Abbildung 5.5: Feature Diagramm des RBPf-SLAM

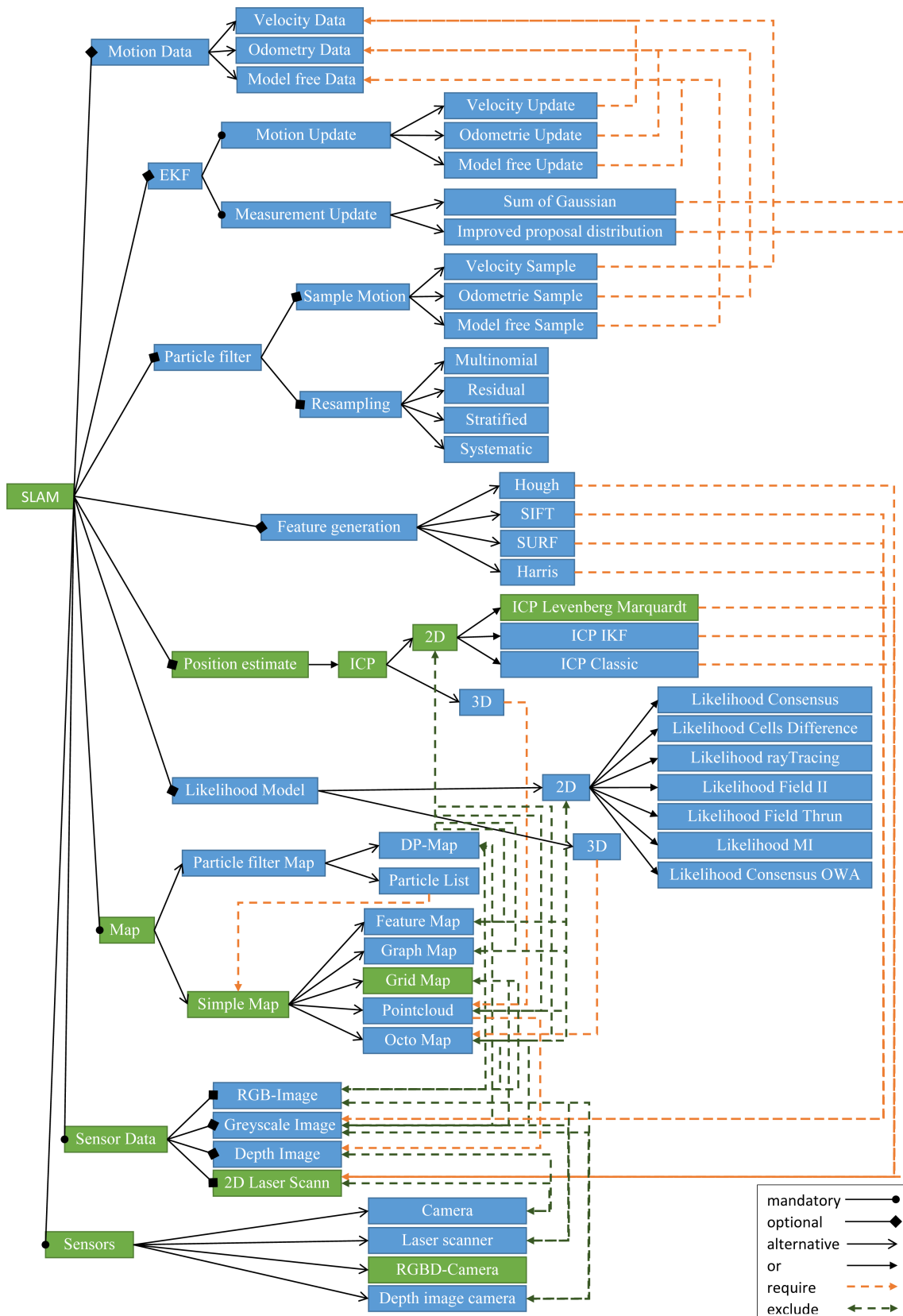


Abbildung 5.6: Feature Diagramm des ICP-SLAM

ICP-SLAM (siehe Abbildung 5.6) unterscheidet sich im Vergleich zu den vorher genannten SLAM-Verfahren darin, dass keine *Motion Data* benötigt werden. Somit fallen alle Features weg, die *Motion Data* voraussetzen, welche das *EKF* sowie das *Particle filter* Feature sind. Wie in dem Diagramm zu erkennen ist, verwendet der ICP-SLAM nur die verpflichtenden Features und die *Position estimate* um mithilfe des *ICP Levenberg Marquardt* Features die Roboterposition zu ermitteln.

Es besteht zwar die Möglichkeit Bewegungsdaten in den SLAM zu integrieren, aber es ist nicht zwingend notwendig, wodurch auch eine solche Konfiguration einen funktionstüchtigen SLAM darstellt.

5.4 ERWEITERUNGSMÖGLICHKEITEN

Das existierende Feature Diagramm kann in weiteren Arbeiten erweitert werden und somit eine größere Anzahl an SLAM-Verfahren abdecken. Wobei meine Übersicht in ihrem Bereich nicht komplett ist und noch um weitere Funktionen bzw. Features erweitert werden kann. Dies könnten u.a. weitere Wahrscheinlichkeitsmodelle sein, die den Bereich der 3-dimensionalen Anwendungen abdeckt. Außerdem fehlen z.B. noch die Feature basierten SLAM-Verfahren, welche im Bereich des Extended Kalman Filters mehrere neue Funktionen zur Folge hätten und noch weitere Feature Detektoren benötigen. Ebenso kann es um weitere spezielle Kartendarstellungen, wie die DP-Map zur Speicheroptimierung, ausgebaut werden.

5.5 ZUSAMMENFASSUNG

Das Ziel dieser Arbeit bestand darin, die existierenden SLAM-Verfahren zu analysieren und die verschiedenen Komponenten zu extrahieren. Im Bereich der Grid Map basierten SLAM-Verfahren wurde die Komponenten Extraktion schrittweise in den Abschnitten 3.1 bis 4 durchgeführt. Aus den Komponenten, ihren Eigenschaften sowie deren Verwendung in den betrachteten SLAM-Verfahren wurde in diesem Abschnitt das Feature Diagramm entwickelt, welches die Zusammenhänge und Abhängigkeiten der verschiedenen Features untereinander beschreibt. Außerdem bekommt der Anwender einen Überblick über die einzelnen Teile der SLAM-Verfahren sowie deren Wirkungsweise untereinander.

Mithilfe der Beispiel SLAM-Verfahren entstand eine leicht lesbare Abbildung zum Verstehen der Arbeitsweise der einzelnen SLAM Implementierungen.

6 ZUSAMMENFASSUNG UND AUSBLICK

Zu Beginn der Arbeit an diesem Thema stand eine große Auswahl von über 30 SLAM-Verfahren, welche für das simultane Lokalisieren und Kartographieren entwickelt wurden. Jedoch existierte keine Übersicht, in der die SLAMs bezüglich ihrer grundlegenden Eigenschaften (z.B. Sensoren, Umgebungen, ...) eingeordnet sind. In Tabelle 3.1 entstand ein Überblick mit dessen Hilfe einem SLAM Entwickler eine erste Filterung anhand der System- und Umgebungskriterien bereitgestellt wird.

Diese Übersicht ist eine Entscheidungshilfe, die dem Softwareentwickler jedoch nicht beim Identifizieren wiederverwendbarer Teilkomponenten hilft. Daraus ergab sich die Frage aus der Einleitung:

1. Besteht die Möglichkeit einzelne Teile des SLAM wiederzuverwenden?

Anhand der rasterbasierten SLAM-Verfahren wurde diese Frage im Abschnitt 3 und 4 beantwortet. Beim Auseinandersetzen mit den Grundlagen des SLAM aus dem Buch von Thrun et al. [TFB06] und dem Analysieren der verschiedenen SLAM-Verfahren mit dem EKF-SLAM, Fast-SLAM usw. entstand der Eindruck, dass diese Verfahren nur in wenige Bereiche unterteilt werden können und nur wenige alternative Methoden existieren. Dies wurde in den weiteren Recherchen mit der großen Anzahl an verschiedenen Verfahren und den damit resultierenden Teilen, die sich in vielen SLAM-Verfahren wiederholen, wie z.B. das Bewegungsmodell oder die Wahrscheinlichkeitsberechnung, widerlegt. Ebenso wurden die Implementierungen einiger SLAM-Verfahren analysiert und es zeigte sich, dass viele Teile mit leicht abgewandelten Implementierungen gleicher Funktionalität existieren.

Das zeigt, dass es Optimierungspotenzial gibt und die Aufsplittung der Verfahren möglich ist. Anhand der Komponenten im Abschnitt 4 ist zu erkennen, dass alle rasterbasierten SLAM-Verfahren aus wenigen Teilen bestehen, von denen wiederum ein Großteil der Komponenten in mehreren SLAMs eingesetzt wird. Mit dieser Erkenntnis kann eine komponentenbasierte Softwareent-

wicklung für SLAM-Verfahren durchgeführt werden. Die Unterteilung in Komponenten wird im MRPT-Framework vorangetrieben, jedoch ist die Mehrzahl aller SLAM Implementierungen nicht nach diesem Konzept entwickelt, wodurch kaum Wiederverwendungsmöglichkeiten existieren. Mithilfe der komponentenbasierten Entwicklung von SLAM-Verfahren könnten Weiterentwicklungen und Verbesserungen einfacher in bestehende SLAMs integriert werden und es ist keine komplette Neuimplementierung nötig. Außerdem besteht Optimierungspotenzial, da Kombinationen verschiedener Funktionen unkomplizierter getestet und neue SLAMs einfacher entwickelt werden können.

Nach der Aufteilung der SLAM-Verfahren in wiederverwendbare Komponenten stellt sich die 2.Frage aus der Einleitung:

2. Besteht die Möglichkeit einzelne Teile des SLAM dynamisch auszutauschen?

Die Austauschbarkeit einzelner Funktionen wurde in Abschnitt 4 gezeigt, indem es mehrere Implementierungen für einzelne Teile gibt, wie z.B. das Resampling als *Multinomial*, *Residual*, *Stratified* oder *Systematic*. Diese Funktionen können zur Laufzeit getauscht werden, da sie die gleichen Parameter besitzen. Eine weitere Frage ist, ob Features an den Knoten des Baumes ebenso austauschbar sind. Der Anwendungsfall besteht darin, einen SLAM während der Kartierung an plötzlich fehlende Bewegungsinformationen anzupassen. Diese Frage kann mit dem Feature Diagramm beantwortet werden.

Das Ziel eines Feature Modells ist es, Produkte aus einer Vielzahl von Möglichkeiten abzuleiten und deren Variabilität darzustellen. Mithilfe des Feature Diagramms sollen die verschiedenen Kombinationsmöglichkeiten und Ausschließungen der Komponenten aus Abschnitt 4 dargestellt werden. Unter Verwendung des Feature Diagramms kann die Frage, ob der SLAM dynamisch auf fehlende Bewegungsdaten angepasst werden kann, mit Hilfe der Core-SLAM Abbildung 5.3 und der ICP-SLAM Abbildung 5.6 beantwortet werden. Wenn bei dem Core-SLAM die Bewegungsdaten wegfallen, muss die Roboterposition auf eine andere Art und Weise ermittelt werden. Hierbei kann als Alternative das *Position estimate* Feature des ICP-SLAM verwendet werden. Mit einer solchen dynamischen Umstellung des Core-SLAM kann der Algorithmus bei Systemänderungen weiterarbeiten.

Außerdem wird dem SLAM Anwender mit dem Feature Diagramm ein weiteres Werkzeug bereitgestellt. Zuvor bestand die Möglichkeit mit der Tabelle 3.1 einen SLAM für ein Anwendungsgebiet auszuwählen, mit dem Feature Diagramm besteht die Möglichkeit einen SLAM zusammenzubauen. Mit den Systemvoraussetzungen (Sensorik, Bewegungsinformationen, ...) können die verfügbaren Komponenten identifiziert und zusammengestellt werden. Somit kann der Entwickler einen SLAM selbst zusammenstellen.

Aufbauend auf den in dieser Arbeit gewonnenen Erkenntnissen können weitere wichtige Probleme analysiert werden. Neben der Möglichkeit des dynamischen Komponentenaustausches wäre es von Interesse, wie viel Speicher oder Rechenleistung die einzelnen Komponenten benötigen bzw. wie genau die jeweiligen Positionsschätzungen sind. Hierfür besteht die Möglichkeit die Komponenten in fortführenden Arbeiten auf ihre Eigenschaften zu prüfen, um SLAM Algorithmen optimaler an ihre Einsatzzwecke anzupassen. Weiterführend kann die Verwendung

verschiedener Komponenten, basierend auf dem Rechenaufwand und der Genauigkeit verschiedener Komponenten, gegenseitig abgewägt werden um dementsprechend den SLAM zusammenzustellen. Ebenfalls kann der Austausch einzelner Komponenten zur Laufzeit zugunsten der Prozessorauslastung oder der Lokalisationsgenauigkeit durchgeführt werden. Dies kann von Interesse sein, wenn der Roboter nicht mehr genügend Akkuleistung besitzt und Rechenleistung zugunsten des Akkus sparen muss.

Der Austausch kann auch andere Gründe haben. Wenn der Roboter z.B. den Core-SLAM verwendet und sich aus einem Raum in einen Korridor bewegt, wäre es hilfreich den SLAM zu wechseln, da der Core-SLAM nicht für Korridore ausgelegt ist. Somit könnten die notwendigen Komponenten ausgetauscht und ein angepasster SLAM eingesetzt werden.

Die Arbeit hat ebenfalls Erweiterungspotenzial hinsichtlich der berücksichtigten SLAM-Verfahren. Momentan werden nur die rasterbasierten SLAMs abgebildet, die auf die Feature basierten SLAMs erweitert werden können. In diesem Zusammenhang würden u.a. weitere Kartentypen und Feature Detektoren hinzukommen. Neue Sensorarten sind ebenfalls vorstellbar, wie z.B. Sensoren, die in den Räumen angebracht sind (Überwachungskameras) und mit deren Hilfe die Roboterposition ermittelt wird.

Im Weiteren besteht die Möglichkeit, dass das erarbeitete Feature Diagramm in einem SLAM Framework umgesetzt oder in andere Plattformen integriert wird. In diesem Zusammenhang könnten die SLAMs der ROS-Plattform durch einen komponentenbasierten Ansatz ersetzt werden, um somit eine Anpassbarkeit des ROS an verschiedene Systemvoraussetzungen und Umgebungen zu ermöglichen. Es besteht ebenso die Möglichkeit das MRPT-Framework weiter auszubauen und um ein Baukastenprinzip, basierend auf den Erkenntnissen dieser Arbeit, zu erweitern. Eine große Anzahl benötigter Komponenten sind bereits implementiert, jedoch ist das Zusammenbauen der SLAM-Verfahren weiterhin mit erheblichem Programmieraufwand verbunden. Eine nützliche Erweiterung wäre die Entwicklung eines Regelwerks zum einfachen Zusammenbauen und dynamischen Anpassen der SLAM-Verfahren.

Abschließend ist festzuhalten, dass die Arbeit einen Einblick in das Potenzial komponentenbasierter SLAM Entwicklung gibt und viele Erweiterungsmöglichkeiten existieren. Aus dieser Arbeit lassen sich viele weiterführende Forschungsarbeiten ableiten.

LITERATURVERZEICHNIS

- [Abb06] ABBEEL, Pieter: *gMapping*. Foliensatz, 2006
- [BFMG08] BLANCO, Jose-Luis ; FERNÁNDEZ-MADRIGAL, Juan-Antonio ; GONZÁLEZ, Javier: Efficient Probabilistic Range-Only SLAM. In: *International Conference on Intelligent Robots and Systems*, 2008
- [BR11] BRUNO, Luigi ; ROBERTSON, Patrick: WiSLAM: improving FootSLAM with WiFi. In: *International Conference on Indoor Positioning and Indoor Navigation*, 2011
- [EHE⁺] ENDRES, Felix ; HESS, Jürgen ; ENGELHARD, Nikolas ; STURM, Jürgen ; BURGARD, Wolfram: Online-6D-SLAM für RGB-D-Sensoren.
- [Eng] ENGELHARDT, Frank: *Umsetzung eines Online-SLAM-Verfahrens auf der Roboterplattform VolksBot-Lab*
- [EP] ELIAZAR, Austin ; PARR, Ronald: DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks.
- [FHC⁺] FOSSEL, Joscha ; HENNES, Daniel ; CLAES, Daniel ; ALERS, Sjriek ; TUYLS, Karl: OctoSLAM: A 3D Mapping Approach to Situational Awareness of Unmanned Aerial Vehicles.
- [FTBD] FOX, Diter ; THRUN, Sebastian ; BURGARD, Wolfram ; DELLAERT, Frank: Particle Filter for Mobile Robot Localization.
- [GCB⁺09] GÉROSSIER, Franck ; CHECCHIN, Paul ; BLANC, Christophe ; CHAPUIS, Roland ; TRASSOUDAIN, Laurent: Trajectory-oriented EKF-SLAM using the Fourier-Mellin Transform applied to Microwave Radar Images. In: *International Conference on Intelligent Robots and Systems*, 2009
- [GEFM12] GUTMANN, Jens-Steffen ; EADE, Ethan ; FONG, Philip ; MUNICH, Mario E.: Vector Field SLAM—Localization by Learning the Spatial Variation of Continuous Signals. In: *Transactions on Robotics*, 2012

- [GKS⁺] GRISSETTI, Giorgio ; KÜMMERLE, Rainer ; STACHNISS, Cyrill ; FRESE, Udo ; HERTZBERG, Christoph: Hierarchical Optimization on Manifolds for Online 2D and 3D Mapping.
- [GSB] GRISSETTI, Giorgio ; STACHNISS, Cyrill ; BURGARD, Wolfram: Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling.
- [HCHX12] HONGJIAN, WANG ; CUN, LI ; HONGLI, LV ; XINGHUA, CHEN: Research on Compressed EKF Based SLAM Algorithm for Unmanned Underwater Vehicle. In: *International IEEE Conference on Intelligent Transportation Systems, 2012*
- [Hor13] HORNING, Armin: *3D Mapping with OctoMap*. Foliensatz, 2013
- [KDa] KAESS, Michael ; DELLAERT, Frank: Covariance Recovery from a Square Root Information Matrix for Data Association.
- [KDb] KWOK, N. M. ; DISSANAYAKE, G.: Bearing-only SLAM in Indoor Environments Using a Modified Particle Filter.
- [KKG⁺] KONOLIGE, Kurt ; GRISSETTI, Giorgio ; KÜMMERLE, Rainer ; BURGARD, Wolfram ; LIMKETKAI, Benson ; VINCENT, Regis: Efficient Sparse Pose Adjustment for 2D Mapping.
- [KGS⁺] KÜMMERLE, Rainer ; GRISSETTI, Giorgio ; STRASDAT, Hauke ; KONOLIGE, Kurt ; BURGARD, Wolfram: g2o: A General Framework for Graph Optimization.
- [KJR⁺] KAESS, Michael ; JOHANNSSON, Hordur ; ROBERTS, Richard ; ILA, Viorela ; LEONARD, John ; DELLAERT, Frank: iSAM2: Incremental Smoothing and Mapping with Fluid Relinearization and Incremental Variable Reordering.
- [KMG⁺] KOHLBRECHER, Stefan ; MEYER, Johannes ; GRABER, Thorsten ; PETERSEN, Karen ; STRYK, Oskar von ; KLINGAUF, Uwe: Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots.
- [KMRD08] KAESS, Michael ; MEMBER, Student ; RANGANATHAN, Ananth ; DELLAERT, Frank: iSAM: Incremental Smoothing and Mapping. In: *Transactions on Robotics, 2008*
- [KRD] KAESS, Michael ; RANGANATHAN, Ananth ; DELLAERT, Frank: Fast Incremental Square Root Information Smoothing.
- [KSC] KERL, Christian ; STURM, Jürgen ; CREMERS, Daniel: Dense Visual SLAM for RGB-D Cameras.
- [mrp] *MRPT-Framework @ONLINE*. <http://www.mrpt.org>
- [MW12] MILFORD, Michael J. ; WYETH, Gordon. F.: SeqSLAM: Visual Route-Based Navigation for Sunny Summer Days and Stormy Winter Nights. In: *International Conference on Intelligent Robots and Systems, 2012*
- [NBN] NIETO, Juan ; BAILEY, Tim ; NEBOT, Eduardo: Scan-SLAM: Combining EKF-SLAM and Scan Correlation.
- [ope] *Open SLAM Plattform @ONLINE*. <http://www.openslam.org>

- [RGS⁺] ROUSSILLON, Cyril ; GONZALEZ, Aurélien ; SOLÀ, Joan ; CODOL, Jean-Marie ; MANSARD, Nicolas ; LACROIX, Simon ; DEVY, Michel: RT-SLAM: a generic and real-time visual SLAM implementation.
- [RKD] RANGANATHAN, Ananth ; KAESS, Michael ; DELLAERT, Frank: Loopy SLAM.
- [ros] ROS Framework @ONLINE. <http://www.ros.org>
- [Sch13] SCHROETER, Julia: *Adaptive Configuration of Cloud-based Applications by Extending Feature-based Product Line Methods*, Fakultät Informatik, Technische Universität Dresden, Diss., 2013
- [SG08] SCHROETER, Christof ; GROSS, Horst-Michael: A sensor-independent approach to RBPf SLAM - Map Match SLAM applied to Visual Mapping. In: *International Conference on Intelligent Robots and Systems*, 2008
- [SH] STEUX, Bruno ; HAMZAOU, Oussama E.: CoreSLAM : a SLAM Algorithm in less than 200 lines of C code.
- [SHT] SEGAL, Aleksandr V. ; HAEHNEL, Dirk ; THRUN, Sebastian: Generalized-ICP.
- [SMNS⁺] SALAS-MORENO, Renato F. ; NEWCOMBE, Richard A. ; STRASDAT, Hauke ; KELLY, Paul H. J. ; DAVISON, Andrew J.: SLAM++: Simultaneous Localisation and Mapping at the Level of Objects.
- [Sta06] STACHNISS, Cyrill: *Exploration and Mapping with Mobile Robots*, Fakultät für Angewandte Wissenschaften, Albert-Ludwigs-Universität Freiburg im Breisgau, Diss., 2006
- [TFB06] THRUN, Sebastian ; FOX, Dieter ; BURGARD, Wolfram: *Probabilistic Robotics*. Cambridge, Massachusetts : The MIT Press., 2006
- [TOA⁺] TIAR, R. ; OUADAH, N. ; AZOUAOU, O. ; DJEHAICH, M. ; ZIANE, H. ; ACHOUR, N.: ICP-SLAM Methods Implementation on a Bi-steerable Mobile Robot.
- [WEL] WALTER, Matthew R. ; EUSTICE, Ryan M. ; LEONARD, John J.: Exactly Sparse Extended Information Filters for Feature-Based SLAM.
- [WHD] WANG, Zhan ; HUANG, Shoudong ; DISSANAYAKE, Gamini: D-SLAM: Decoupled Localization and Mapping for Autonomous Robots.
- [ZHD] ZHAO, Liang ; HUANG, Shoudong ; DISSANAYAKE, Gamini: Linear SLAM: A Linear Solution to the Feature-based and Pose Graph SLAM based on Submap Joining.